

Call-by-value, call-by-name and the vectorial behaviour of algebraic λ -calculus

Alejandro Díaz-Caro¹, Simon Perdrix^{2,1}, Christine Tasson³, and
Benoît Valiron⁴

¹ LIG, Université de Grenoble, France
{alejandro.diaz-caro,simon.perdrix}@imag.fr

² CNRS, France

³ PPS, Université Paris Diderot, France
christine.tasson@pps.jussieu.fr

⁴ University of Pennsylvania, USA
benoit.valiron@monoidal.net

Abstract. We examine the relationship between the *algebraic λ -calculus*, a fragment of the differential λ -calculus and the *linear-algebraic λ -calculus*, a candidate λ -calculus for quantum computation. Both calculi are algebraic: each one is equipped with an additive and a scalar-multiplicative structure, and their set of terms is closed under linear combinations. However, the two languages were built using different approaches: the former is a call-by-name language whereas the latter is call-by-value; the former considers algebraic equalities whereas the latter approaches them through rewrite rules.

In this paper, we analyse how these different approaches relate one to the other. To this end, we propose four canonical languages based on each of the possible choices: call-by-name versus call-by-value, algebraic equality versus algebraic rewriting. We show that the various languages simulate one another. Due to subtle interaction between beta-reduction and algebraic rewriting, to make the languages consistent some additional hypotheses such as confluence might be required. We carefully devise the required properties for each proof, making them general enough to be valid for any sub-language satisfying the corresponding properties.

1 Introduction

Algebraic λ -calculi. Two algebraic versions of the λ -calculus arise independently in distinct contexts: the algebraic λ -calculus (λ_{alg}) [26] and the linear algebraic λ -calculus (λ_{lin}) [6]. The former has been introduced in the context of linear logic as a fragment of the differential λ -calculus [14]: the algebraic structure allows to gather in a non deterministic manner different terms, *i.e.* each term represents one possible execution. The latter has been introduced as a candidate λ -calculus for quantum computation: in λ_{lin} , a linear combination of terms reflects the phenomenon of superposition, *i.e.* the capability for a quantum system to be in two or more states at the same time.

Four languages with different behaviours. In both languages, functions which are linear combinations of terms are interpreted pointwise: $(\alpha.f + \beta.g) x = \alpha.(f) x + \beta.(g) x$, where “.” denotes the scalar multiplication. The two languages differ on the treatment of the arguments. In λ_{lin} , in order to deal with the algebraic structure, any function is considered as a linear map: $(f) (\alpha.x + \beta.y) \rightarrow^* \alpha.(f) x + \beta.(f) y$, reflecting the fact that any quantum evolution is a linear map. It reflects a call-by-value behaviour in the sense that the argument is evaluated until one has a base term. In the opposite, λ_{alg} has a call-by-name evolution: $(\lambda x M) N \rightarrow M[x := N]$, without any restriction on N . As a consequence, the evolutions are different as illustrated by the following example. In λ_{lin} , $(\lambda x (x) x) (\alpha.y + \beta.z) \rightarrow^* \alpha.(y) y + \beta.(z) z$ while in λ_{alg} , $(\lambda x (x) x) (\alpha.y + \beta.z) \rightarrow (\alpha.y + \beta.z) (\alpha.y + \beta.z) = \alpha^2.(y) y + \alpha\beta.(y) z + \beta\alpha.(z) y + \beta^2.(z) z$.

Because they were designed for different purposes, another difference appears between the two languages: the way the algebraic part of the calculus is treated. In λ_{lin} , the algebraic structure is captured with a rewrite system, whereas in λ_{alg} terms are considered up to algebraic equivalence.

The two choices – call-by-value versus call-by-name, algebraic equality versus algebraic reduction – allow one to construct four possible calculi. We name them $\lambda_{lin}^{\rightarrow}$, $\lambda_{lin}^{\leftarrow}$, $\lambda_{alg}^{\rightarrow}$, and $\lambda_{alg}^{\leftarrow}$, see Figure 1 where they are presented according to their evolution policy and the way they take care of the algebraic part of the language.

Inspired by λ_{lin} and λ_{alg} , the operational semantics of the four languages we introduce slightly differ from the original ones for focusing on the particularities of the calculi we are interested in: reduction strategy and handling of algebraic structure in *programs*.

A first modification is that in all four languages, we avoid reduction under lambda abstractions. As a consequence, contrary to λ_{alg} , the λ -abstraction is not linear anymore: $\lambda x (\alpha.M + \beta.N) \neq \alpha.\lambda x N + \beta.\lambda x N$. This restriction is a common restriction: reducing under λ could be considered as “optimising the program”. Also note that since we consider $\lambda_{alg}^{\rightarrow}$ with algebraic rewrite rules instead of the equalities used in λ_{alg} , we need two extra rules: $\alpha.M + M \rightarrow (\alpha + 1).M$ and $M + M \rightarrow (1 + 1).M$. These rules were not needed with equalities, since $M = 1.M$.

Concerning $\lambda_{lin}^{\rightarrow}$ and $\lambda_{lin}^{\leftarrow}$, restrictions originally imposed in λ_{lin} on the rewrite system to ensure confluence are replaced by restrictions which make $\lambda_{lin}^{\rightarrow}$ and $\lambda_{lin}^{\leftarrow}$ actual call-by-value languages. For example, the rule $(M + N) L \rightarrow (M) L + (N) L$ when $M + N$ is closed-normal form is replaced by only asking L to be a value. Notice that even in the original language λ_{lin} , waiving the restrictions makes sense when confluence can be ensured by other means, see *e.g.* [3, 22].

Contribution: relation between the four languages through simulation. Although these languages behave differently, we show in this paper that they simulate each other. This result connects works done in linear logic [11–14, 16, 17, 20, 25] and works on quantum computation [2–5, 7, 10, 22, 24].

We show that call-by-value algebraic λ -calculi simulate call-by-name ones and *vice versa* by extending the continuation passing style (CPS) [18] to the algebraic case. We also provide simulations between algebraic equality and alge-

braic reduction in both directions. The simulations we prove are summed up in Figure 2. The solid arrows stand for theorems that do not require confluence in their hypothesis whereas the dashed arrows stand for theorems that do.

A preliminary version of this work was presented in [9].

Consistency. Without restrictions on the set of terms, both algebraic reductions and algebraic equalities cause problems of consistency, albeit differently.

Let $Y_M = (\lambda x (M + (x) x)) \lambda x (M + (x) x)$. In a system with algebraic reduction, the term $Y_M - Y_M$ reduces to 0, but also reduces to $M + Y_M - Y_M$ and hence to M , breaking confluence. To solve this issue, several distinct techniques can be used to make an algebraic calculus confluent. In [6], restrictions on reduction rules are introduced, e.g. $\alpha.M + \beta.M \rightarrow (\alpha + \beta).M$ if M is closed normal. In [3, 4, 7, 22, 24], type systems are set up to forbid diverging terms such as Y_M .

In a system with algebraic equalities, if M and N are any terms, the term M reduces to $M + Y_{N-M} - Y_{N-M}$, therefore to N . In λ_{alg} a restriction to positive scalars is proposed to solve the problem. However such a solution does not work in a system with algebraic reduction (*cf.* Section 3).

In this paper we do not make a choice *a priori*, instead we show that the simulations between the four calculi are correct, providing a general enough methodology to work in a large variety of restrictions on the language. Therefore, we do not force one specific method to make the calculi consistent, leaving the choice to the reader.

Plan of the paper. In Section 2, we define the set of terms and the rewrite systems we consider in the paper. In Section 3, we discuss the confluence of the algebraic rewrite systems. Section 4 is concerned with the actual simulations. In Section 4.1 we consider the correspondence between algebraic reduction and algebraic equality whereas in Section 4.2 and 4.3 we consider the distinction call-by-name versus call-by-value. In Section 4.4, we show how the simulations can compose to obtain the correspondence between any two of the four languages. In Section 5 we conclude by providing some paths for future work. Most of the omitted and sketched proofs are fully developed in the appendix.

2 Algebraic λ -calculi

The languages λ_{lin} and λ_{alg} share the same syntax, defined as follows:

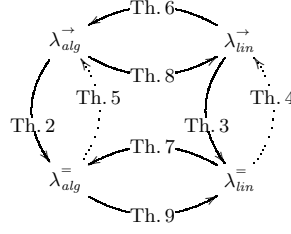
$$\begin{aligned} M, N, L &::= V \mid (M) N \mid \alpha.M \mid M + N && \text{(terms),} \\ U, V, W &::= 0 \mid B \mid \alpha.V \mid V + W && \text{(values),} \\ B &::= x \mid \lambda x M && \text{(basis terms),} \end{aligned}$$

where α ranges over a ring, the *ring of scalars*. We use the notation $M - N$ as a shorthand for $M + (-1).N$. Note that we could have asked for a semiring instead; in fact we shall see in Section 3.2 that the analysis we develop here can be adapted to semirings of scalars.

	call-by-name	call-by-value
algebraic reduction	$\lambda_{alg}^{\rightarrow}$	$\lambda_{lin}^{\rightarrow}$
algebraic equality	λ_{alg}^{\equiv}	λ_{lin}^{\equiv}

cf. Definition 1

Fig. 1. The four algebraic λ -calculi.



$A \rightarrow B$ means “ A is simulated by B ”

Fig. 2. Relations between the languages.

We provide a complete formalisation of the rewrite rules and show how they relate to each other. We summarise in Figure 3 all the rewrite rules that are being used.

The rules are grouped with respect to their intuitive meaning. We use the usual notation regarding rewrite systems: Given a rewrite system R , we write R^* for its reflexive and transitive closure. That is, xR^*y is valid if $y = x$ or if there exists a rewrite sequence $xRx_1R\cdots Rx_nRy$ linking x and y . We write R_{\leftrightarrow} for the symmetric closure of R , that is, the relation that satisfies $xR_{\leftrightarrow}y$ if and only if xRy or yRx .

The original languages λ_{lin} and λ_{alg} made particular assumptions both on the reduction strategy and the handling of algebraic structure under the reduction. In this paper, we consider separately the distinction call-by-name/call-by-value and the distinction algebraic equality/algebraic reduction. We develop therefore four languages: a call-by-value language λ_{lin}^{\equiv} with algebraic equality, a call-by-value language $\lambda_{lin}^{\rightarrow}$ with algebraic reduction, a call-by-name language λ_{alg}^{\equiv} with algebraic equality and a call-by-name language $\lambda_{alg}^{\rightarrow}$ with algebraic reduction.

These four languages are summarised in Figure 1.

Definition 1. We use the following notations for the rewrite systems obtained by combining the rules described in Figure 3:

$$\begin{aligned}
\rightarrow_a &:= A \cup L \cup \xi & \rightarrow_\ell &:= A_l \cup A_r \cup L \cup \xi \cup \xi_{\lambda_{lin}} & \rightarrow_{\beta_v} &:= \beta_v \cup \xi \cup \xi_{\lambda_{lin}} \\
\rightarrow_a^{\equiv} &:= (\rightarrow_a)_{\leftrightarrow} & \rightarrow_\ell^{\equiv} &:= (\rightarrow_\ell)_{\leftrightarrow} & \rightarrow_{\beta_n} &:= \beta_n \cup \xi
\end{aligned}$$

We define the following four languages and their associated rewrite systems:

Language	Corresponding Rewrite System
$\lambda_{lin}^{\rightarrow}$	$\rightarrow_{\ell \cup \beta} := (\rightarrow_\ell) \cup (\rightarrow_{\beta_v})$
λ_{lin}^{\equiv}	$\rightarrow_{\ell \cup \beta}^{\equiv} := (\rightarrow_\ell^{\equiv}) \cup (\rightarrow_{\beta_v})$
$\lambda_{alg}^{\rightarrow}$	$\rightarrow_{a \cup \beta} := (\rightarrow_a) \cup (\rightarrow_{\beta_n})$
λ_{alg}^{\equiv}	$\rightarrow_{a \cup \beta}^{\equiv} := (\rightarrow_a^{\equiv}) \cup (\rightarrow_{\beta_n})$

3 Discussion on consistency and confluence

3.1 Local confluence

In this section we show that the four languages $\lambda_{lin}^{\rightarrow}$, $\lambda_{alg}^{\rightarrow}$, λ_{lin}^{\equiv} , and λ_{alg}^{\equiv} are locally confluent. A rewrite system R is locally confluent if whenever xRy and

SPECIFIC RULES FOR $\lambda_{alg}^{\rightarrow}$ AND $\lambda_{alg}^{\leftarrow}$	
Call-by-name (β_n)	LINEARITY OF THE APPLICATION (A)
$(\lambda x M) N \rightarrow M[x := N]$	$(M + N) L \rightarrow (M) L + (N) L$ $(\alpha.M) N \rightarrow \alpha.(M) N$ $(0) M \rightarrow 0$
SPECIFIC RULES FOR $\lambda_{lin}^{\rightarrow}$ AND $\lambda_{lin}^{\leftarrow}$	
Call-by-value (β_v)	CONTEXT RULE ($\xi_{\lambda_{lin}}$)
$(\lambda x M) B \rightarrow M[x := B]$	$\frac{M \rightarrow M'}{(V) M \rightarrow (V) M'}$
LINEARITY OF THE APPLICATION	
Left linearity (A_l)	Right linearity (A_r)
$(M + N) V \rightarrow (M) V + (N) V$ $(\alpha.M) V \rightarrow \alpha.(M) V$ $(0) V \rightarrow 0$	$(B) (M + N) \rightarrow (B) M + (B) N$ $(B) (\alpha.M) \rightarrow \alpha.(B) M$ $(B) 0 \rightarrow 0$
COMMON RULES	
RING RULES ($L = Asso \cup Com \cup F \cup S$)	
Associativity ($Asso$)	Commutativity (Com)
$M + (N + L) \rightarrow (M + N) + L$ $(M + N) + L \rightarrow M + (N + L)$	$M + N \rightarrow N + M$
Factorization (F)	Simplification (S)
$\alpha.M + \beta.M \rightarrow (\alpha + \beta).M$ $\alpha.M + M \rightarrow (\alpha + 1).M$ $M + M \rightarrow (1 + 1).M$ $\alpha.(\beta.M) \rightarrow (\alpha\beta).M$	$\alpha.(M + N) \rightarrow \alpha.M + \alpha.N$ $1.M \rightarrow M$ $0.M \rightarrow 0$ $\alpha.0 \rightarrow 0$ $0 + M \rightarrow M$
CONTEXT RULES (ξ)	
$\frac{M \rightarrow M'}{(M) N \rightarrow (M') N} \quad \frac{M \rightarrow M'}{M + N \rightarrow M' + N}$	$\frac{N \rightarrow N'}{M + N \rightarrow M + N'} \quad \frac{M \rightarrow M'}{\alpha.M \rightarrow \alpha.M'}$

Fig. 3. Rewrite rules with U, V and W values, B a basis term, M, M', N, N' and L any terms.

xRz there is t such that yR^*t and zR^*t . In comparison, a rewrite system R is confluent if whenever xR^*y and xR^*z there is t such that yR^*t and zR^*t . Notice that confluence implies local confluent whereas the inverse is not true.

We first concentrate on the algebraic rules. For each of these calculi, we use the reductions describing the algebraic structure: \rightarrow_a and \rightarrow_ℓ correspond to an oriented rewriting description whereas $\rightarrow_a^{\leftarrow}$ and $\rightarrow_\ell^{\leftarrow}$ correspond to a description by equalities (since every rewrite rule can be reversed, cf. Definition 1).

Lemma 1. *The rewrite systems $\rightarrow_a, \rightarrow_\ell, \rightarrow_a^{\leftarrow}$ and $\rightarrow_\ell^{\leftarrow}$ are locally confluent.*

Proof. For \rightarrow_ℓ and \rightarrow_a , we give a semi-automatised proof in the interactive theorem prover Coq [8]. The interested reader can find the proof in [23] which is sketched in the appendix. Since for any rewrite system R , its symmetric closure R_{\leftrightarrow} is trivially locally confluent, both \rightarrow_a^\equiv and \rightarrow_ℓ^\equiv are locally confluent. \square

The rewrites systems considered in this paper are also locally confluent in the presence of the β -rewrite rules.

Lemma 2 (Local confluence). *The four languages in Figure 1 are locally confluent.*

Proof (Sketch). The local confluence of the algebraic fragment is proven in Lemma 1. The beta-reduction is confluent using a straightforward extension of the confluence of lambda calculus. Finally, the beta-reduction and the algebraic fragments commute, making each rewrite system locally confluent. \square

3.2 Simulations and the confluence issue

In this section, we show that the algebraic fragments are confluent modulo associativity and commutativity. Concerning the full languages, we show that they are either not confluent or trivially confluent (in the sense that any term is reducing to any other). As a consequence, we introduce a generic notion of language fragment to describe confluent and consistent sub-languages. In particular, fragments are used in simulations theorems in Section 4 for abstractly representing confluent sub-languages.

The algebraic fragment. It is clear that neither \rightarrow_a nor \rightarrow_ℓ is strongly normalising: with both systems one can go back and forth between $M + N$ and $N + M$. They are however strongly normalising “modulo associativity and commutativity” in the sense that any rewrite sequence consists eventually of terms that are equal modulo associativity and commutativity. On the contrary, the rewrite systems \rightarrow_a^\equiv and \rightarrow_ℓ^\equiv are not.

In order to formalise this, let us denote AC the system generated by $AC = Asso \cup Com$ and \overline{R} the rewrite system obtained by taking off the rules $Asso$ and Com where R stands for \rightarrow_a or \rightarrow_ℓ . Hence, \overrightarrow{a} stands for the system generated by $A \cup S \cup F \cup \xi$ and $\overrightarrow{\ell}$ for the system generated by $A_l \cup A_r \cup S \cup F \cup \xi \cup \xi_{lin}$.

Definition 2. *Let R be either \rightarrow_ℓ or \rightarrow_a and $M_1 R M_2 R \dots$ be a reduction sequence (finite or not) characterised by the set of terms $\{M_i\}_i$ and the set of rules $\{R_i\}_i$ used to go from M_i to M_{i+1} , where R_i stands for a fixed rule in R . We say that the reduction is AC-finite if the set of indices i such that $R_i \in \overline{R}$ is finite. The AC-length of the rewrite sequence is the cardinal of this set of indices. The rewrite system R is AC-strongly-normalising (AC-SN) if for any term M , there exists a number n such that the AC-length of any rewrite sequence starting at M is less than n . A term M is AC-normal with respect to a rewrite system R if any rewrite sequence starting with M consists only of rules AC .*

Theorem 1. *The systems \rightarrow_a and \rightarrow_ℓ are AC-SN.*

Proof. We use the technique described in [6]. An auxiliary measure is defined on terms by $|(M) N| = (3|M| + 2)(3|N| + 2)$, $|\alpha.M| = 1 + 2|M|$, $|M + N| = 2 + |M| + |N|$, $|0| = 0$, $|\lambda x.M| = 1$ and $|x| = 1$. This measure is preserved by rules AC and strictly decreasing on the other algebraic rules. \square

Local confluence plus strong normalisation implies confluence (see for example [21]).

Corollary 1. *The rewrite systems \rightarrow_a and \rightarrow_ℓ are confluent, modulo AC.* \square

Although we have proved that the four languages under consideration are locally confluent, neither $\lambda_{lin}^\rightarrow$ nor $\lambda_{alg}^\rightarrow$ is confluent: In each one, the term $Y_M - Y_M$ rewrites both to 0 and M , where $Y_M = (\lambda x (M + (x) x)) \lambda x (M + (x) x)$.

Regarding λ_{lin}^\leftarrow and λ_{alg}^\leftarrow , without restriction both are trivially confluent since for all terms M and N , M reduces to N : $M = M + Y_{N-M} - Y_{N-M} \rightarrow N$. Hence, with the algebraic equality, both languages can simulate any rewrite system.

For getting back consistency, it is of course possible to modify the rewrite systems as in [6] but it would break the correspondence between call-by-value and call-by-name. In this paper we propose instead to restrict the set of terms. In the literature, there have been two methods:

With algebraic equalities, Vaux [26] considers non-negative scalars (semiring) on a language with algebraic equality. The restriction on scalars is enough for getting unicity of normal forms. Although this solves the consistency problem for the languages with algebraic equality, it does not give confluence for the languages with algebraic reduction. Indeed, consider the critical pair $Y_M + Y_M \rightarrow_\ell 2.Y_M$, $Y_M + Y_M \rightarrow_{\beta_v} Y_M + M + Y_M \rightarrow_\ell 2.Y_M + M$. The term $2.Y_M$ can only produce an even number of M 's: we cannot close the pair.

With algebraic reductions, other papers [3, 4, 7] use type systems for retrieving strong normalisation on a language with algebraic reduction. This technique could be directly adapted to our setting since it is possible to have subject reduction in these cases.

The simulations theorems that we develop in this paper are correct in a general untyped setting (and in fact trivially true when we simulate a language with algebraic reduction with a language with algebraic equality as remarked above), but also true if one restrict the scalars to a semiring (as done in [26]), or if we restrict the terms to any typed setting, provided that the languages $\lambda_{lin}^\rightarrow$ and $\lambda_{alg}^\rightarrow$ satisfy subject reduction and that the CPS translations preserve typability. Thus, in this paper we do not restrict the calculi *a priori*, instead, we propose a notion of *language fragments* to parametrise the simulation results. The definition of fragment is general enough to capture many settings: various typed systems, but also the restrictions to a given set of terms such as the set of AC-SN terms or taking scalars from a semiring.

We define formally a fragment in the following way:

Definition 3. *A fragment S of $\lambda_{lin}^\rightarrow$ (resp. $\lambda_{alg}^\rightarrow$) is a language defined on a subset of terms closed under $\rightarrow_{\ell \cup \beta}$ -reduction (resp. $\rightarrow_{a \cup \beta}$ -reduction). The rewrite system of S is inherited from the one of $\lambda_{lin}^\rightarrow$ (resp. $\lambda_{alg}^\rightarrow$).*

The definition of a fragment in the presence of algebraic equalities should be treated carefully. Indeed, note that the algebraic equalities are defined as $M \rightarrow^= N$ if and only if $M \rightarrow N$ or $N \rightarrow M$. As a consequence, for any subset S of terms closed under $\rightarrow^=$ -reduction, if M is in S then for any N (in S or not), $M + N - N \in S$ since $M \rightarrow^= M + N - N$. We therefore need to define the algebraic equality with respect to the particular subset of terms under consideration.

Definition 4. A fragment S of $\lambda_{lin}^=$ (resp. $\lambda_{alg}^=$) is a fragment of $\lambda_{lin}^{\rightarrow}$ (resp. $\lambda_{alg}^{\rightarrow}$) together with an algebraic equality defined as $M \rightarrow_{\ell}^=^S N$ (resp. $M \rightarrow_a^=^S N$) if and only if $M, N \in S$ and $N \rightarrow_{\ell} M$ or $M \rightarrow_{\ell} N$ (resp. $N \rightarrow_a M$ or $M \rightarrow_a N$). The β -reduction is not modified.

Convention 1 When referring to a fragment of $\lambda_{lin}^=$ (resp. $\lambda_{alg}^=$), we use the abuse of notation $\rightarrow_{\ell}^=$ (resp. $\rightarrow_a^=$) instead of $\rightarrow_{\ell}^=^S$ (resp. $\rightarrow_a^=^S$) for the restricted rewrite system, when the fragment under consideration is clear.

4 Simulations

The core of the paper is concerned with the mutual simulations of the four languages.

The first class of problems relates algebraic reduction with algebraic equality. If simulating a language with algebraic reduction with a language with algebraic equality is not specially difficult, going in the opposite direction is not possible in general. Indeed, if $0 =_{\ell} Y_M - Y_M \rightarrow_{\beta_v} Y_M + M - Y_M =_{\ell} M$ is possible in $\lambda_{lin}^=$, (where $Y_M = (\lambda x (M + (x) x)) \lambda x (M + (x) x)$) it is difficult to see how one could make 0 go to M in $\lambda_{lin}^{\rightarrow}$ without further hypotheses. In this section, we show that a fragment of a language with algebraic equality can be simulated by the corresponding fragment with algebraic reduction provided that the latter is *confluent* (Theorems 4 and 5).

The second class of problems is concerned with call-by-value and call-by-name. In this paper, the simulations of call-by-name by call-by-value and its reverse are treated using continuation passing style (CPS), extending the techniques described in [15, 18] to the algebraic case (Theorems 6, 7, 8 and 9).

The results are summarised in Figure 2. Solid arrows correspond to results where no particular hypothesis on the language is made. Dashed arrows correspond to results where confluence is required.

4.1 Algebraic reduction versus algebraic equality

As the relation $\rightarrow_{\ell \cup \beta}$ is contained in $\rightarrow_{\ell \cup \beta}^=$ and the relation $\rightarrow_{a \cup \beta}$ is contained in $\rightarrow_{a \cup \beta}^=$, the first simulation theorems are trivial.

Theorem 2. For any term M if $M \rightarrow_{a \cup \beta} N$, then $M \rightarrow_{a \cup \beta}^= N$. □

Theorem 3. For any term M if $M \rightarrow_{\ell \cup \beta} N$, then $M \rightarrow_{\ell \cup \beta}^= N$. □

The simulations going in the other direction are only valid in the presence of confluence. In the following two theorems, the algebraic equality is defined with respect to the considered fragment (see Definition 4.)

Theorem 4. *For any term M in a confluent fragment of $\lambda_{lin}^{\rightarrow}$, if $M \rightarrow_{\ell \cup \beta}^* V$, then $M \rightarrow_{\ell \cup \beta}^* V'$, with $V \rightarrow_{\ell}^* V'$.*

Proof. First note that a value can only reduce to another value. This follows by direct inspection of the rewriting rules. We proceed by induction on the length of the reduction.

- If $M \rightarrow_{\ell \cup \beta}^* M$, then choose $V' = M$ and note that $M \rightarrow_{\ell \cup \beta}^* M$.
- Assume the result true for $M \rightarrow_{\ell \cup \beta}^* V$: there is a value V' such that $M \rightarrow_{\ell \cup \beta}^* V'$ and $V \rightarrow_{\ell}^* V'$. Let $N \rightarrow_{\ell \cup \beta}^* M$. Case distinction:
 - $N \rightarrow_{\beta_v} M$, then $N \rightarrow_{\beta_v} M \rightarrow_{\ell \cup \beta}^* V'$ which implies $N \rightarrow_{\ell \cup \beta}^* V'$.
 - $N \rightarrow_{\ell}^* M$, then either $N \rightarrow_{\ell} M$, and then this case is analogous to the previous one, or $M \rightarrow_{\ell} N$. Due to the confluence of the subset, there exists a term L such that $N \rightarrow_{\ell \cup \beta}^* L$ and $V' \rightarrow_{\ell}^* L$, implying that L is a value, thus $V' \rightarrow_{\ell}^* L$. Then we have $V' \rightarrow_{\ell}^* L$ and $V \rightarrow_{\ell}^* V'$, so $L \rightarrow_{\ell}^* V$, closing the case. \square

Theorem 5. *For any term M in a confluent fragment of $\lambda_{alg}^{\rightarrow}$, if $M \rightarrow_{a \cup \beta}^* V$, then $M \rightarrow_{a \cup \beta}^* V'$, with $V \rightarrow_a^* V'$.*

Proof. Similar to the previous theorem. \square

4.2 Call-by-name simulates call-by-value

To prove the simulation of $\lambda_{lin}^{\rightarrow}$ with $\lambda_{alg}^{\rightarrow}$ and the simulation of $\lambda_{lin}^{\leftarrow}$ with $\lambda_{alg}^{\leftarrow}$, we introduce an algebraic extension of the continuation passing style used to prove that call-by-name simulates call-by-value in the regular λ -calculus [18].

Let $\llbracket \cdot \rrbracket : \Lambda_{\lambda_{lin}} \rightarrow \Lambda_{\lambda_{alg}}$ be the following encoding where f, g and h are fresh variables.

$$\begin{aligned} \llbracket x \rrbracket &= \lambda f (f) \ x, & \llbracket 0 \rrbracket &= 0, \\ \llbracket \lambda x M \rrbracket &= \lambda f (f) \ \lambda x \llbracket M \rrbracket, & \llbracket (M) \ N \rrbracket &= \lambda f (\llbracket M \rrbracket) \ \lambda g (\llbracket N \rrbracket) \ \lambda h ((g) \ h) \ f, \\ \llbracket \alpha.M \rrbracket &= \lambda f (\alpha. \llbracket M \rrbracket) \ f, & \llbracket M + N \rrbracket &= \lambda f (\llbracket M \rrbracket + \llbracket N \rrbracket) \ f. \end{aligned}$$

Let Ψ be the encoding for values defined by $\Psi(x) = x$, $\Psi(0) = 0$, $\Psi(\lambda x M) = \lambda x \llbracket M \rrbracket$, $\Psi(\alpha.V) = \alpha. \Psi(V)$, $\Psi(V + W) = \Psi(V) + \Psi(W)$. Note that this encoding is compatible with substitution (proof by induction on M):

Lemma 3. $\llbracket M[x := B] \rrbracket = \llbracket M \rrbracket[x := \Psi(B)]$ with B a base term. \square

Using this encoding, we can simulate $\lambda_{lin}^{\rightarrow}$ with $\lambda_{alg}^{\rightarrow}$, as formalised in the following theorem. The sketch of the proof is developed in the second part of this section.

Theorem 6 (Simulation). *For any term M if $M \rightarrow_{\ell \cup \beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$.*

Example 1. For any terms M and N , let $\langle M, N \rangle := \lambda y ((y) M) N$. Let **copy** be the term $\lambda x \langle x, x \rangle$, and let $U = \lambda x N_1$ and $V = \lambda x N_2$ be two values. Then $(\text{copy}) (U + V) \rightarrow_{\ell \cup \beta}^* \langle U, U \rangle + \langle V, V \rangle$ and $(\text{copy}) (U + V) \rightarrow_{a \cup \beta}^* \langle U + V, U + V \rangle$. We consider the simulation $\lambda_{in}^{\rightarrow}$ to $\lambda_{alg}^{\rightarrow}$. The translation $\llbracket (\text{copy}) (U + V) \rrbracket$ is $\lambda f (\llbracket \text{copy} \rrbracket) \lambda g (\llbracket U + V \rrbracket) \lambda h ((g) h) f$, where $\llbracket \text{copy} \rrbracket$ is the term $\lambda f (f) \lambda x \llbracket \langle x, x \rangle \rrbracket$, with $\llbracket \langle M, N \rangle \rrbracket$ being $\lambda f (f) \Psi(\langle M, N \rangle)$, $\llbracket U + V \rrbracket$ being $\lambda f (\llbracket U \rrbracket + \llbracket V \rrbracket) f$, and $\llbracket U \rrbracket$ being $\lambda g (g) \Psi(U)$. We now rewrite $M = (\llbracket \text{copy} \rrbracket) (U + V) \rrbracket \lambda z z$ in $\lambda_{alg}^{\rightarrow}$.

$$\begin{aligned}
M &\rightarrow_{a \cup \beta} (\llbracket \text{copy} \rrbracket) \lambda g (\llbracket U + V \rrbracket) \lambda h ((g) h) \lambda z z \\
&= (\lambda f (f) \lambda x \llbracket \langle x, x \rangle \rrbracket) \lambda g (\llbracket U + V \rrbracket) \lambda h ((g) h) \lambda z z \\
&\rightarrow_{a \cup \beta} (\lambda g (\llbracket U + V \rrbracket) \lambda h ((g) h) \lambda z z) \lambda x \llbracket \langle x, x \rangle \rrbracket \\
&\rightarrow_{a \cup \beta} (\llbracket U + V \rrbracket) \lambda h ((\lambda x \llbracket \langle x, x \rangle \rrbracket) h) \lambda z z \\
&\rightarrow_{a \cup \beta} (\llbracket U \rrbracket + \llbracket V \rrbracket) \lambda h ((\lambda x \llbracket \langle x, x \rangle \rrbracket) h) \lambda z z \\
&\rightarrow_{a \cup \beta} (\llbracket U \rrbracket) \lambda h ((\lambda x \llbracket \langle x, x \rangle \rrbracket) h) \lambda z z + (\llbracket V \rrbracket) \lambda h ((\lambda x \llbracket \langle x, x \rangle \rrbracket) h) \lambda z z \\
&\rightarrow_{a \cup \beta}^* (\lambda h ((\lambda x \llbracket \langle x, x \rangle \rrbracket) h) \lambda z z) \Psi(U) + (\lambda h ((\lambda x \llbracket \langle x, x \rangle \rrbracket) h) \lambda z z) \Psi(V) (*) \\
&\rightarrow_{a \cup \beta}^* ((\lambda x \llbracket \langle x, x \rangle \rrbracket) \Psi(U)) \lambda z z + ((\lambda x \llbracket \langle x, x \rangle \rrbracket) \Psi(V)) \lambda z z \\
&\rightarrow_{a \cup \beta}^* (\llbracket \langle x, x \rangle \rrbracket [x := \Psi(U)]) \lambda z z + (\llbracket \langle x, x \rangle \rrbracket [x := \Psi(V)]) \lambda z z \\
&\rightarrow_{a \cup \beta}^* (\llbracket \langle U, U \rangle \rrbracket) \lambda z z + (\llbracket \langle V, V \rangle \rrbracket) \lambda z z \quad (\text{Lemma 3}) \\
&\rightarrow_{a \cup \beta}^* (\lambda z z) \Psi(\langle U, U \rangle) + (\lambda z z) \Psi(\langle V, V \rangle) \quad (**) \\
&\rightarrow_{a \cup \beta}^* \Psi(\langle U, U \rangle) + \Psi(\langle V, V \rangle) \\
&= \Psi(\langle U, U \rangle + \langle V, V \rangle)
\end{aligned}$$

Similarly, one can relate fragments of $\lambda_{alg}^{\rightarrow}$ to fragments of $\lambda_{in}^{\rightarrow}$ as follows.

Theorem 7 (Simulation). *For any two fragments S_ℓ of $\lambda_{in}^{\rightarrow}$ and S_a of $\lambda_{alg}^{\rightarrow}$ such that $\forall M \in S_\ell, (\llbracket M \rrbracket) \lambda x x \in S_a$, and for any term M in S_ℓ , if $M \rightarrow_{\ell \cup \beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$.*

Again, the sketch of the proof is developed later in the section.

Remark 1. As we already noted several times in this paper, without restricting the languages, Theorem 7 would be trivial. Any term reducing to any other one, the desired reduction would be of course valid without restriction. This theorem shows that if the calculi are restricted to fragments, the result is still true. One example of such fragments is found by taking the restriction of scalars to non-negative elements, as in [26].

Once a term is encoded it can be reduced either by $\rightarrow_{a \cup \beta}^*$ or by $\rightarrow_{\ell \cup \beta}^*$ (respectively $\rightarrow_{a \cup \beta}^*$ or $\rightarrow_{\ell \cup \beta}^*$) without distinction, and still obtain the same result. We state this fact as a corollary:

Corollary 2 (Indifference). *(1) For any term M , if $M \rightarrow_{\ell \cup \beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* \Psi(V)$; (2) For any fragment S of $\lambda_{in}^{\rightarrow}$ such that $\forall M \in S, (\llbracket M \rrbracket) \lambda x x \in S$, and for any term M in S , if $M \rightarrow_{\ell \cup \beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* \Psi(V)$.*

Proof. It suffices to check the proofs of Theorems 6 and 7 to verify that all the reductions $\rightarrow_{a\cup\beta}^*$ are done by rules common in both languages. \square

Example 2. Note that in Example 1 one could have as well rewrite with $\rightarrow_{\ell\cup\beta}$ which illustrates the indifference property (Corollary 2).

Now we proceed to prove Theorems 6 and 7. The proof in [18] can be extended to the algebraic case.

An administrative operation. We define a convenient infix operation $(:)$ capturing the behaviour of translated terms. For example, if B is a base term, *i.e.* a variable or an abstraction, then its translation into $\lambda_{alg}^{\rightarrow}$ is $\llbracket B \rrbracket = \lambda f \ (f) \ \Psi(B)$. If we apply this translated term to a certain K , we obtain $(\lambda f \ (f) \ \Psi(B)) \ K \rightarrow_{a\cup\beta} (K) \ \Psi(B)$. We define $B : K = (K) \ \Psi(B)$ and get that $(\llbracket B \rrbracket) \ K \rightarrow_{a\cup\beta} B : K$. This fact will be generalised to $(\llbracket M \rrbracket) \ K \rightarrow_{a\cup\beta} M : K$ in Lemma 4.

Definition 5. Let $(:): A_{\lambda_{lin}} \times A_{\lambda_{alg}} \rightarrow A_{\lambda_{alg}}$ be the infix binary operation defined by:

$$\begin{array}{ll} 0 : K = 0 & (0) \ N : K = 0 \\ B : K = (K) \ \Psi(B) & (B) \ N : K = N : \lambda f \ ((\Psi(B)) \ f) \ K \\ \alpha.M : K = \alpha.(M : K) & (\alpha.M) \ N : K = \alpha.(M) \ N : K \\ M + N : K = M : K + N : K & (M + N) \ L : K = ((M) \ L + (N) \ L) : K \\ & ((M) \ N) \ L : K = (M) \ N : \lambda g \ (\llbracket L \rrbracket) \ \lambda h \ ((g) \ h) \ K \end{array}$$

Lemma 4. If K is a base term, then for any M , $(\llbracket M \rrbracket) \ K \rightarrow_{a\cup\beta}^* M : K$.

Proof. Structural induction on M . We give the case $M = (M') \ N$, as an example. First an intermediate result is needed: for any M , $M : \lambda g \ (\llbracket N \rrbracket) \ \lambda h \ ((g) \ h) \ K \rightarrow_{a\cup\beta} (M) \ N : K$. This can be proved by structural induction on M .

Then $(\llbracket (M') \ N \rrbracket) \ K = (\lambda f \ (\llbracket M' \rrbracket) \ \lambda g \ (\llbracket N \rrbracket) \ \lambda h \ ((g) \ h) \ f) \ K$ which $\rightarrow_{a\cup\beta}$ -reduces to $(\llbracket M' \rrbracket) \ \lambda g \ (\llbracket N \rrbracket) \ \lambda h \ ((g) \ h) \ K$. Note that $\lambda g \ (\llbracket N \rrbracket) \ \lambda h \ ((g) \ h) \ K$ is a base term, so by the induction hypothesis the above term reduces to $M' : \lambda g \ (\llbracket N \rrbracket) \ \lambda h \ ((g) \ h) \ K$ which by the previous intermediate result, $\rightarrow_{a\cup\beta}$ -reduces to $(M') \ N : K$. \square

The following lemmas and its corollary state that the $(:)$ operation preserves reduction.

Lemma 5. If $M \rightarrow_{\ell} N$ then $\forall K$ base term, $M : K \rightarrow_a^* N : K$.

Proof. Induction on the possible rule applied from $M \rightarrow_{\ell} N$. We give one simple case as an example. Let $\alpha.(M + N) \rightarrow_{\ell} \alpha.M + \alpha.N$. Then $\alpha.(M + N) : K = \alpha.(M : K + N : K) \rightarrow_a \alpha.(M : K) + \alpha.(N : K) = \alpha.M : K + \alpha.N : K$. \square

Lemma 6. If $M \rightarrow_{\ell\cup\beta} N$ then $\forall K$ base term, $M : K \rightarrow_{a\cup\beta}^* N : K$.

Proof. If $M \rightarrow_{\ell} N$, then use Lemma 5. If $M \rightarrow_{\beta_v} N$, then we prove it by induction on the possible rule applied (either β_v , ξ_{lin} or one of ξ). We give the case of the β_v -reduction as an example: $(\lambda x \ M) \ B : K = B : \lambda f \ ((\Psi(\lambda x \ M)) \ f) \ K = (\lambda f \ ((\Psi(\lambda x \ M)) \ f) \ K) \ \Psi(B)$, β_n -reducing to the term $((\Psi(\lambda x \ M)) \ \Psi(B)) \ K = ((\lambda x \ \llbracket M \rrbracket) \ \Psi(B)) \ K$ which β_n -reduces to $\llbracket M \rrbracket[x := \Psi(B)] \ K$, equal by Lemma 3 to $\llbracket M[x := B] \rrbracket \ K$. By Lemma 4, it $\rightarrow_{a\cup\beta}^*$ -reduces to $M[x := B] : K$. \square

Corollary 3. *If $M \rightarrow_{\ell \cup \beta}^* N$ then $\forall K$ base terms, $M : K \rightarrow_{a \cup \beta}^* N : K$.*

Proof. By case distinction. If $M \rightarrow_{\ell \cup \beta} N$, then by Lemma 6, $M : K \rightarrow_{a \cup \beta}^* N : K$, which implies $M : K \rightarrow_{a \cup \beta}^* N : K$. If $N \rightarrow_{\ell} M$, then by Lemma 5, $N : K \rightarrow_a^* M : K$, which also implies $M : K \rightarrow_{a \cup \beta}^* N : K$. \square

Finally, the $(:)$ operation also captures the translation of values in the following way:

Lemma 7. *For any value V , $V : \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$*

Proof. We proceed by structural induction on V . Let V be a base term. Then $V : \lambda x x = (\lambda x x) \Psi(V) \rightarrow_{a \cup \beta} \Psi(V)$. Let $V = V_1 + V_2$. Then $V : \lambda x x = V_1 : \lambda x x + V_2 : \lambda x x$, which by the induction hypothesis, reduces to $\Psi(V_1) + \Psi(V_2) = \Psi(V)$. Let $V = \alpha.V'$. Then $V : \lambda x x = \alpha.(V' : \lambda x x)$, which by the induction hypothesis, reduces to $\alpha.\Psi(V') = \Psi(V)$. Let $V = 0$. Then $V : \lambda x x = 0 = \Psi(V)$. \square

Example 3. We discuss Example 1 in the light of these results. The term $(*)$ is equal to the terms $(\text{copy})(U + V) : \lambda z.z$ and $((\text{copy})U + (\text{copy})V) : \lambda z.z$. The term $(**)$ is equal to the term $(\langle U, U \rangle + \langle V, V \rangle) : \lambda z.z$ which reduces to $\Psi(\langle U, U \rangle + \langle V, V \rangle)$. So we do have the rewrites requested by Lemmas 4, 6 and 7.

Now the proofs of Theorems 6 and 7 go as follows.

Proof (Proof of Theorem 6). From Lemma 4, $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* M : \lambda x x$ and from Lemma 6, it $\rightarrow_{a \cup \beta}^*$ -reduces to $V : \lambda x x$. From Lemma 7, $V : \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$. \square

Proof (Proof of Theorem 7). From Lemma 4, $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* M : \lambda x x$, and this implies that $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* M : \lambda x x$. From Corollary 3, this latter term $\rightarrow_{a \cup \beta}^*$ -reduces to $V : \lambda x x$. From Lemma 7, $V : \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$, which implies that $V : \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$. Note that since $(\llbracket M \rrbracket) \lambda x x \in S_a$, $M : \lambda x x$ is also in S_ℓ due to the closeness under \rightarrow_a^* of S_a . The same applies to $M : \lambda x x$, thus also to $V : \lambda x x$ and finally to $\Psi(V)$. \square

4.3 Call-by-value simulates call-by-name

The simulation of $\lambda_{alg}^\rightarrow$ with $\lambda_{lin}^\rightarrow$. To state that $\lambda_{lin}^\rightarrow$ simulates $\lambda_{alg}^\rightarrow$, we use an algebraic extension of the continuation passing style encoding following again [18].

Let $\llbracket \cdot \rrbracket : \lambda_{alg}^\rightarrow \rightarrow \lambda_{lin}^\rightarrow$ be the following encoding where f, g and h are fresh variables.

$$\begin{aligned} \llbracket x \rrbracket &= x, & \llbracket 0 \rrbracket &= \lambda f (0) f, \\ \llbracket \lambda x M \rrbracket &= \lambda f (f) \lambda x \llbracket M \rrbracket, & \llbracket (M) N \rrbracket &= \lambda f (\llbracket M \rrbracket) \lambda g ((g) \llbracket N \rrbracket) f, \\ \llbracket \alpha.M \rrbracket &= \lambda f (\alpha. \llbracket M \rrbracket) f, & \llbracket M + N \rrbracket &= \lambda f (\llbracket M \rrbracket + \llbracket N \rrbracket) f. \end{aligned}$$

This encoding satisfies two useful properties (the first is a trivial result and the second follows by induction on M).

Lemma 8. For all terms M , the term $\llbracket M \rrbracket$ is a base term. \square

Lemma 9. $\llbracket M[x := N] \rrbracket = \llbracket M \rrbracket[x := \llbracket N \rrbracket]$. \square

Let Φ be the encoding for values defined by $\Phi(x) = (x) \lambda y y$, $\Phi(0) = 0$, $\Phi(\lambda x M) = \lambda x \llbracket M \rrbracket$, $\Phi(\alpha.V) = \alpha.\Phi(V)$, $\Phi(V + W) = \Phi(V) + \Phi(W)$. Simulation theorems, similar to Theorems 6 and 7, can be stated as follows.

Theorem 8 (Simulation). For any program M (i.e. closed term), if $M \rightarrow_{a\cup\beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell\cup\beta}^* \Phi(V)$.

Theorem 9 (Simulation). For any two fragments S_a of λ_{alg}^- and S_ℓ of λ_{lin}^- such that $\forall M \in S_a$, $(\llbracket M \rrbracket) \lambda x x \in S_\ell$, and for any program M in S_a , if $M \rightarrow_{a\cup\beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell\cup\beta}^* \Phi(V)$.

A result similar to Corollary 2 can also be formulated. It is proven in a similar manner.

Corollary 4 (Indifference). (1) For any program M , if $M \rightarrow_{a\cup\beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a\cup\beta}^* \Phi(V)$; (2) For any fragment S of λ_{alg}^- such that $\forall M \in S$, $(\llbracket M \rrbracket) \lambda x x \in S$, and for any program M in S , if $M \rightarrow_{a\cup\beta}^* V$ where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a\cup\beta}^* \Phi(V)$. \square

Before moving to the description of the proof of Theorems 8 and 9, let us consider an example.

Example 4. We illustrate Theorem 8 using the term $(\text{copy}) (U + V)$ of Example 1 which reduces to $\langle U, U \rangle + \langle V, V \rangle$ in λ_{lin}^- and to $\langle U + V, U + V \rangle$ in λ_{alg}^- . The translation $\llbracket (\text{copy}) (U + V) \rrbracket$ is the term $\lambda f (\llbracket \text{copy} \rrbracket) \lambda g ((g) \llbracket U + V \rrbracket) f$, where $\llbracket \text{copy} \rrbracket$ is $\lambda f (f) \lambda x \llbracket \langle x, x \rangle \rrbracket$. $\llbracket \langle M, N \rangle \rrbracket$ is $\lambda f (f) \Phi(\langle M, N \rangle)$, $\llbracket U + V \rrbracket$ is $\lambda f (\llbracket U \rrbracket + \llbracket V \rrbracket) f$ and $\llbracket U \rrbracket$ is $\lambda g (g) \Phi(U)$. We now rewrite $N = (\llbracket (\text{copy}) (U + V) \rrbracket) \lambda z z$ in λ_{alg}^- .

$$\begin{aligned}
N &\rightarrow_{\ell\cup\beta} (\llbracket \text{copy} \rrbracket) \lambda g ((g) \llbracket U + V \rrbracket) \lambda z z \\
&= (\lambda f (f) \lambda x \llbracket \langle x, x \rangle \rrbracket) \lambda g ((g) \llbracket U + V \rrbracket) \lambda z z \\
&\rightarrow_{\ell\cup\beta} (\lambda g ((g) \llbracket U + V \rrbracket) \lambda z z) \lambda x \llbracket \langle x, x \rangle \rrbracket \\
&\rightarrow_{\ell\cup\beta} ((\lambda x \llbracket \langle x, x \rangle \rrbracket) \llbracket U + V \rrbracket) \lambda z z & (***) \\
&\text{(Lemma 8)} \rightarrow_{\ell\cup\beta} (\llbracket \langle x, x \rangle \rrbracket [x := \llbracket U + V \rrbracket]) \lambda z z \\
&\text{(Lemma 9)} = (\llbracket \langle U + V, U + V \rangle \rrbracket) \lambda z z \\
&\rightarrow_{\ell\cup\beta} (\lambda z z) \Phi(\langle U + V, U + V \rangle) & (****) \\
&\rightarrow_{\ell\cup\beta} \Phi(\langle U + V, U + V \rangle)
\end{aligned}$$

Proof of the simulation theorems. In Section 4.2, the proofs of the simulations theorems were performed using an administrative operation “ \cdot ” and three intermediate results, as follows (the term K is taken as a base term). (1) Prove that $(\llbracket M \rrbracket) K \rightarrow_{a\cup\beta}^* M : K$; (2) prove that if $M \rightarrow_{\ell\cup\beta} N$ then $M : K \rightarrow_{a\cup\beta}^* N : K$; (3) prove that if V is a value, $V : \lambda x. x \rightarrow_{a\cup\beta}^* \Psi(V)$. For the simulation theorems of the present section, we use a similar procedure.

An administrative operation. We keep the same notation for the administrative, infix operation defined for the purpose of the proof.

Definition 6. Let $(\cdot) : \Lambda_{\lambda_{alg}} \times \Lambda_{\lambda_{lin}} \rightarrow \Lambda_{\lambda_{lin}}$ be the infix binary operation defined by:

$$\begin{array}{ll} 0 : K = 0 & (0) N : K = 0 \\ B : K = (K) \Phi(B) & (B) N : K = ((\Phi(B)) \llbracket N \rrbracket) K \\ \alpha.M : K = \alpha.(M : K) & (\alpha.M) N : K = \alpha.(M) N : K \\ M + N : K = M : K + N : K & (M + N) L : K = ((M) L + (N) L) : K \\ & ((M) N) L : K = (M) N : \lambda f ((f) \llbracket L \rrbracket) K \end{array}$$

The three lemmas needed for the proof of the simulation theorems now read as follow.

Lemma 10. If K is a base term, for any closed term M $(\llbracket M \rrbracket) K \rightarrow_{\ell \cup \beta}^* M : K$.

Proof. The proof is done by structural induction on M . We follow the sketch of the proof of Lemma 4, and give the case $M = (M') N$, as an example. First we prove by induction on M that $M : \lambda g ((g) \llbracket N \rrbracket) K \rightarrow_{\ell \cup \beta}^* (M) N : K$. Then $(\llbracket M' \rrbracket) K = (\lambda f (\llbracket M' \rrbracket) \lambda g ((g) \llbracket N \rrbracket) f) K \rightarrow_{\ell \cup \beta} (\llbracket M' \rrbracket) \lambda g ((g) \llbracket N \rrbracket) K$. Note that $\lambda g ((g) \llbracket N \rrbracket) K$ is a base term, so by the induction hypothesis the above term reduces to $M' : \lambda g ((g) \llbracket N \rrbracket) K$ which by the previous intermediate result, $\rightarrow_{\ell \cup \beta}$ -reduces to $(M') N : K$. \square

Lemma 11. If $M \rightarrow_{a \cup \beta} N$ then $\forall K$ base term, $M : K \rightarrow_{\ell \cup \beta}^* N : K$

Proof. Case by case on the rules of $\lambda_{alg}^{\rightarrow}$. We give the case of the β_n -reduction as an example: $(\lambda x M) N : K = ((\Phi(\lambda x M)) \llbracket N \rrbracket) K = ((\lambda x \llbracket M \rrbracket) \llbracket N \rrbracket) K$ which by Lemma 8, $\rightarrow_{\ell \cup \beta}$ -reduces to $(\llbracket M \rrbracket[x := \llbracket N \rrbracket]) K$. This, by Lemma 9, is equal to $(\llbracket M[x := N] \rrbracket) K$ and this, by Lemma 10, $\rightarrow_{\ell \cup \beta}^*$ -reduces to $M[x := N] : K$. Note that in the previous derivation, the reduction $(\lambda x \llbracket M \rrbracket) \llbracket N \rrbracket \rightarrow_{\ell \cup \beta} \llbracket M \rrbracket[x := \llbracket N \rrbracket]$ is valid since for any term N , $\llbracket N \rrbracket$ is a base term. \square

Lemma 12. If V is a value and K is a base term, $V : \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V)$. \square

Example 5. We discuss Example 4 in the light of these results. The term $(***)$ is equal to the terms $(\text{copy}) (U + V) : \lambda z.z$. The term $(****)$ is equal to the term $\langle U + V, U + V \rangle : \lambda z.z$ which reduces to $\Phi(\langle U + V, U + V \rangle)$. Again, we have the rewrites requested by Lemmas 10, 11 and 12.

We are now ready to prove the simulation theorems. As advertised, these proofs reflect the exact same structures of the proofs of Theorems 6 and 7.

Proof (Theorem 8). From Lemma 10, $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* M : \lambda x x$, from Lemma 11 it $\rightarrow_{\ell \cup \beta}^*$ -reduces to $V : \lambda x x$. From Lemma 12, $V : \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V)$. \square

Proof (Theorem 9). From Lemma 10, $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* M : \lambda x x$, and this implies that $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* M : \lambda x x$. A result equivalent to Corollary 3 can be shown as easily: if $M \rightarrow_{\bar{a}}^* N$ then for all base terms K , $M : K \rightarrow_{\bar{\ell}}^* N : K$. This entails that $M : \lambda x x \rightarrow_{\ell \cup \beta}^* N : \lambda x x$. From Lemma 12, $V : \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V)$, which implies that $V : \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V)$. Note that since $(\llbracket M \rrbracket) \lambda x x \in S_{\ell}$, $M : \lambda x x$ is also in S_{ℓ} due to the closeness under $\rightarrow_{\bar{\ell}}$ of S_{ℓ} . The same applies to $M : \lambda x x$, thus also to $V : \lambda x x$ and finally to $\Phi(V)$. \square

4.4 The remaining simulations

In Figure 2, some arrows are missing. We are now showing that the already existing arrows “compose” well. The first two simulations are $\lambda_{alg}^{\rightarrow} \rightarrow \lambda_{lin}^{\leftarrow}$ and $\lambda_{lin}^{\rightarrow} \rightarrow \lambda_{alg}^{\leftarrow}$ and do not require confluence.

Theorem 10. *For any program M , if $M \rightarrow_{\ell \cup \beta}^* V$ (resp. $M \rightarrow_{a \cup \beta}^* V$) where V is a value, then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$ (resp. $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V)$).*

Proof. Given that $M \rightarrow_{\ell \cup \beta}^* V$, by Theorem 6, $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$, which by Theorem 2 implies $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V)$. Analogously, given that $M \rightarrow_{a \cup \beta}^* V$, by Theorem 8, $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V)$, which by Theorem 3 implies that $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V)$. \square

The other two simulations are $\lambda_{alg}^{\leftarrow} \rightarrow \lambda_{lin}^{\rightarrow}$ and $\lambda_{lin}^{\leftarrow} \rightarrow \lambda_{alg}^{\rightarrow}$ and they do require confluence.

Theorem 11. *For any program M in a confluent fragment of $\lambda_{lin}^{\leftarrow}$ (resp. $\lambda_{alg}^{\leftarrow}$), if $M \rightarrow_{\ell \cup \beta}^* V$ (respectively $M \rightarrow_{a \cup \beta}^* V$) then $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V')$ with $V \rightarrow_{\ell}^* V'$ (respectively $(\llbracket M \rrbracket) \lambda x x \rightarrow_{\ell \cup \beta}^* \Phi(V')$ with $V \rightarrow_a^* V'$).*

Proof. Given that $M \rightarrow_{\ell \cup \beta}^* V$ and that M is in a confluent fragment, Theorem 4 states that $M \rightarrow_{\ell \cup \beta}^* V'$ with $V \rightarrow_{\ell}^* V'$. In addition, Theorem 6 states that $(\llbracket M \rrbracket) \lambda x x \rightarrow_{a \cup \beta}^* \Psi(V')$. The other result is similar using Theorems 5 and 8. \square

5 Conclusion and perspectives

In this paper we described four canonical algebraic lambda-calculi with vectorial structures, recapitulating the few existing means of writing such a language. We show how each language can simulate the other, by taking care of marking where confluence is used or not.

As already shown by Plotkin [18], if the simulation of call-by-value by call-by-name is sound, it fails to be complete for general (possibly non-terminating) programs. A known solution to this problem is developed in [19]. Recent work [1] shown that the technique can be adapted to the algebraic case to retrieve completeness. The work [19] develop a Galois connection between call-by-name and call-by-value. A direction for study is to build on this work to also get a Galois connection in the algebraic case.

Concerning semantics, the algebraic λ -calculus admits finiteness spaces as a model [12, 13]. What is the structure of the model of the linear algebraic λ -calculus induced by the continuation-passing style translation in finiteness spaces? The algebraic lambda-calculus can be equipped with a differential operator. What is the corresponding operator in call-by-value through the translation?

Acknowledgements Many thanks to Pablo Arrighi, Ali Assaf and Lionel Vaux for fruitful discussions and suggestions. This work is supported by the CNRS - INS2I PEPS project QuAND.

References

1. A. Assaf and S. Perdrix. Completeness of algebraic CPS simulations. To appear in *Proc. DCM'11*. <http://membres-lig.imag.fr/perdrix/cps-completeness.html>
2. T. Altenkirch and J. J. Grattage. A functional quantum programming language. In *Proc. LICS'05*, pp 249–258.
3. P. Arrighi and A. Díaz-Caro. Scalar system F for linear-algebraic λ -calculus: Towards a quantum physical logic. In *Proc. QPL'09*, vol. 270-2 of *ENTCS*, pp 219–229.
4. P. Arrighi, A. Díaz-Caro, and B. Valiron. A type system for the vectorial aspects of the linear-algebraic lambda-calculus. To appear in *Proc. DCM'11*. <http://membres-liglab.imag.fr/diazcaro/vectorial.pdf>
5. P. Arrighi and G. Dowek. A computational definition of the notion of vectorial space. In *Proc. WRLA'04*, vol. 117 of *ENTCS*, pp. 249–261.
6. P. Arrighi and G. Dowek. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In *Proc. RTA'08*, vol. 5117 of *LNCS*, pp. 17–31.
7. P. Buiras, A. Díaz-Caro, and M. Jaskelioff. Lower bounds for scalars in a typed algebraic λ -calculus. Preprint at arXiv:1102.0749, February 2011.
8. Coq Dev. Team. *The Coq proof assistant reference manual*. INRIA, 8.2 edition.
9. A. Díaz-Caro, S. Perdrix, C. Tasson, and B. Valiron. Equivalence of algebraic λ -calculi. In *Informal pro. HOR'10*, pp 6–11.
10. A. Díaz-Caro and B. Petit. Sums in linear algebraic lambda-calculus. Preprint at arXiv:1011.3542, November 2010.
11. T. Ehrhard. On Köthe sequence spaces and linear logic. *MSCS*, 12(5):579–623, 2003.
12. T. Ehrhard. Finiteness spaces. *MSCS*, 15(4):615–646, 2005.
13. T. Ehrhard. A finiteness structure on resource terms. In *Proc. LICS'10*, pp. 402–410.
14. T. Ehrhard and L. Regnier. The differential λ -calculus. *TCS*, 309(1):1–41, 2003.
15. M. J. Fischer. λ -calculus schemata. *ACM SIGPLAN Notice*, 7(1):104–109, 1972.
16. M. Pagani and S. Ronchi Della Rocca. Solvability in resource lambda calculus. In *Proc. FOSSACS'10*, vol. 6014 of *LNCS*, pp. 358–373.
17. M. Pagani and P. Tranquilli. Parallel reduction in resource lambda-calculus. In *Proc. APLAS'09*, vol. 5904 of *LNCS*, pp. 226–242.
18. G.D. Plotkin. Call by name, call by value and the λ -calculus. *TCS*, 1(2):125–159, 1975.
19. A. Sabry and P. Wadler. A reflexion on call-by-value. *TOPLAS* 19(6):916–941, 1997.
20. C. Tasson. Algebraic totality, towards completeness. In *Proc. TLCA'09*, vol. 5608 of *LNCS*, pp 325–340.
21. TeReSe. *Term Rewriting Systems*. Cambridge University Press, 2003.
22. B. Valiron. Semantics of a typed algebraic lambda-calculus. In *Proc. DCM'10*, vol. 26 of *EPTCS*, pp. 147–158.
23. B. Valiron. Coq proof. <http://www.monoidal.net/vectorial-alglin-coqproof-v2.tgz>.
24. B. Valiron. Orthogonality and algebraic λ -calculus. In *Proc. QPL'10*, pp. 169–175.
25. L. Vaux. On linear combinations of λ -terms. In *Proc. RTA'07*, vol. 4533 of *LNCS*, pp. 374–388.
26. L. Vaux. The algebraic lambda calculus. *MSCS*, 19(5):1029–1059, 2009.

A Omitted proofs

A.1 Proof of Lemma 2

Proof.

- If $M \rightarrow N$ and $M \rightarrow N'$, using only algebraic rules, then this has been already proven in Lemma 1.
- If $M \rightarrow N$ and $M \rightarrow N'$, using only beta-reduction, this is a trivial extension of the confluence of lambda calculus.
- If $M \rightarrow N$ by an algebraic rule and $M \rightarrow N'$ by beta reduction, then in $\lambda_{lin}^{\rightarrow}$ a term of the form $(\lambda x M') B$ has to be a subterm of M , since M beta-reduces. Note that M' cannot reduce since it is under a lambda and B cannot reduce since it is a base term. Then the beta-reduction and the algebraic-reduction are independent in $\lambda_{lin}^{\rightarrow}$, and so this result is trivial. In $\lambda_{alg}^{\rightarrow}$ a term of the form $(\lambda x M') N$ has to be a subterm of M . Note that M' cannot reduce since it is under a lambda and N cannot reduce since it is an argument. Then again the beta-reduction and the algebraic-reduction are independent in $\lambda_{alg}^{\rightarrow}$, and so this result is trivial.

A.2 Proof of Lemma 3

Proof. Structural induction on M .

- $M = x$. Cases:
 - $B = y$. Then $M[x := B] = y$, and so $\llbracket M[x := B] \rrbracket = \lambda f (f) y = \lambda f (f) x[y/x] = \llbracket M \rrbracket[x := \Psi(B)]$.
 - $B = \lambda y N$. Then $\llbracket M[x := B] \rrbracket = \lambda f (f) \lambda y \llbracket N \rrbracket = \lambda f (f) x[\lambda y \llbracket N \rrbracket / x] = \llbracket M \rrbracket[x := \Psi(B)]$.
- $M = y$. Then $\llbracket M[x := B] \rrbracket = \llbracket M \rrbracket[x := \Psi(B)] = \llbracket M \rrbracket$.
- $M = 0$. Analogous to previous case.
- $M = \lambda y N$. Then

$$\begin{aligned}
 \llbracket (\lambda y N)[x := B] \rrbracket &= \llbracket \lambda y (N[x := B]) \rrbracket \\
 &= \lambda f (f) \lambda y \llbracket N[x := B] \rrbracket \\
 &\quad \text{by the induction hypothesis} \\
 &= \lambda f (f) \lambda y \llbracket N \rrbracket[x := \Psi(B)] \\
 &= (\lambda f (f) \lambda y \llbracket N \rrbracket)[x := \Psi(B)] \\
 &= \llbracket M \rrbracket[x := \Psi(B)]
 \end{aligned}$$

- $M = (N_1) N_2$. Then

$$\begin{aligned}
 \llbracket M[x := B] \rrbracket &= \llbracket ((N_1) N_2)[x := B] \rrbracket \\
 &= \llbracket (N_1[x := B]) N_2[x := B] \rrbracket \\
 &= \lambda f (\llbracket N_1[x := B] \rrbracket) \lambda g (\llbracket N_2[x := B] \rrbracket) \lambda h ((g) h) f
 \end{aligned}$$

$$\begin{aligned}
& \text{by the induction hypothesis} \\
& = \lambda f (\llbracket N_1 \rrbracket [x := \Psi(B)]) \lambda g (\llbracket N_2 \rrbracket [x := \Psi(B)]) \lambda h ((g) h) f \\
& = \lambda f (\llbracket N_1 \rrbracket) \lambda g (\llbracket N_2 \rrbracket) \lambda h ((g) h) f [x := \Psi(B)] \\
& = \llbracket (N_1) N_2 \rrbracket [x := \Psi(B)] \\
& = \llbracket M \rrbracket [x := \Psi(B)]
\end{aligned}$$

– $M = \alpha.N$. Then

$$\begin{aligned}
\llbracket M[x := B] \rrbracket &= \llbracket (\alpha.N)[x := B] \rrbracket \\
&= \llbracket \alpha.(N[x := B]) \rrbracket \\
&= \lambda f (\alpha.\llbracket N[x := B] \rrbracket f) \\
&\quad \text{by the induction hypothesis} \\
&= \lambda f (\alpha.\llbracket N \rrbracket [x := \Psi(B)] f) \\
&= (\lambda f (\alpha.\llbracket N \rrbracket f)) [x := \Psi(B)] \\
&= \llbracket \alpha.N \rrbracket [x := \Psi(B)] \\
&= \llbracket M \rrbracket [x := \Psi(B)]
\end{aligned}$$

– $M = N_1 + N_2$. Then

$$\begin{aligned}
\llbracket M[x := B] \rrbracket &= \llbracket (N_1 + N_2)[x := B] \rrbracket \\
&= \llbracket N_1[x := B] + N_2[x := B] \rrbracket \\
&= \lambda f ((\llbracket N_1[x := B] \rrbracket) + \llbracket N_2[x := B] \rrbracket) f) \\
&\quad \text{by the induction hypothesis} \\
&= \lambda f ((\llbracket N_1 \rrbracket [x := \Psi(B)] + \llbracket N_2 \rrbracket [x := \Psi(B)])) f) \\
&= (\lambda f ((\llbracket N_1 \rrbracket + \llbracket N_2 \rrbracket) f)) [x := \Psi(B)] \\
&= \llbracket N_1 + N_2 \rrbracket [x := \Psi(B)] \\
&= \llbracket M \rrbracket [x := \Psi(B)]
\end{aligned}$$

A.3 Proof of Lemma 4

Proof. Structural induction on M .

- $M = x$. Then $(\llbracket x \rrbracket) K = (\lambda f (f) x) K \rightarrow_{a \cup \beta} (K) x = x : K$.
- $M = \lambda x N$. Then $(\llbracket \lambda x N \rrbracket) K = (\lambda f (f) \lambda x \llbracket N \rrbracket) K$ and by definition of Ψ this is equal to $(\lambda f (f) \Psi(\lambda x N)) K \rightarrow_{a \cup \beta} (K) \Psi(\lambda x N) = \lambda x N : K$.
- $M = 0$. Then $(\llbracket 0 \rrbracket) K = (0) K \rightarrow_{a \cup \beta} 0 = 0 : K$.
- $M = M' + N$. Then $(\llbracket M' + N \rrbracket) K = (\lambda f (\llbracket M' \rrbracket + \llbracket N \rrbracket) f) K$ which $\rightarrow_{a \cup \beta}$ -reduces to $(\llbracket M' \rrbracket + \llbracket N \rrbracket) K \rightarrow_{a \cup \beta} (\llbracket M' \rrbracket) K + (\llbracket N \rrbracket) K$ which $\rightarrow_{a \cup \beta}$ -reduces by the induction hypothesis to $M' : K + N : K = M' + N : K$.
- $M = \alpha.N$. Then $(\llbracket \alpha.N \rrbracket) K = (\lambda f (\alpha.\llbracket N \rrbracket) f) K \rightarrow_{a \cup \beta} (\alpha.\llbracket N \rrbracket) K$ which $\rightarrow_{a \cup \beta}$ -reduces to $\alpha.(\llbracket N \rrbracket) K$ and this, by the induction hypothesis, $\rightarrow_{a \cup \beta}$ -reduces to $\alpha.(N : K) = \alpha.N : K$.

- $M = (M') N$. Then $(\llbracket (M') N \rrbracket) K = (\lambda f (\llbracket M' \rrbracket) \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) f) K$ which $\rightarrow_{a \cup \beta}$ -reduces to $(\llbracket M' \rrbracket) \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K$. Notice that the term $\lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K$ is a value, so by the induction hypothesis the above term reduces to $M' : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K$. We do a second induction, over M' , to prove that $M' : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K \rightarrow_{a \cup \beta} (M') N : K$.
 - If $M' = (M_1) M_2$, then $M' : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K = ((M_1) M_2) N : K = (M') N : K$.
 - If M' is a base term, then $M' : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K$ is equal to $(\lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K) \Psi(M') \rightarrow_{a \cup \beta} (\llbracket N \rrbracket) \lambda h ((\Psi(M')) h) K$ which by the main induction hypothesis $\rightarrow_{a \cup \beta}$ -reduces to $N : \lambda h ((\Psi(M')) h) K$, and this is equal to $(M') N : K$.
 - If $M' = \alpha.M_1$, then the term $M' : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K$ is equal to $\alpha.M_1 : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K = \alpha.(M_1 : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K)$ which by the second induction hypothesis $\rightarrow_{a \cup \beta}$ -reduces to $\alpha.((M_1) N : K) = (\alpha.M_1) N : K = (M') N : K$.
 - If $M' = M_1 + M_2$, then $M' : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K = M_1 + M_2 : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K$ which is equal to $M_1 : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K + M_2 : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K$ which $\rightarrow_{a \cup \beta}$ -reduces by the second induction hypothesis to $(M_1) N : K + (M_2) N : K = (M_1 + M_2) N : K = (M') N : K$.
 - If $M' = 0$ then $M : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K = 0 : \lambda g (\llbracket N \rrbracket) \lambda h ((g) h) K = 0 = (0) N : K = (M') N : K$.

A.4 Proof of Lemma 5

Proof. Case by case on the rules \rightarrow_ℓ .

Rules A_r

- $(B) (M + N) \rightarrow_\ell (B) M + (B) N$, with B being a base term. Then $(B) (M + N) : K = M + N : \lambda f ((\Psi(B)) f) K = M : \lambda f ((\Psi(B)) f) K + N : \lambda f ((\Psi(B)) f) K = (B) M : K + (B) N : K = (B) M + (B) N : K$.
- $(B) \alpha.M \rightarrow_\ell \alpha.(B) M$, with B base term. Then $(B) \alpha.M : K = \alpha.M : \lambda f ((\Psi(B)) f) K = \alpha.(M : \lambda f ((\Psi(B)) f) K) = \alpha.((B) M : K) = \alpha.(B) M : K$.
- $(B) 0 \rightarrow_\ell 0$, with B a base term. Then $(B) 0 : K = 0 : \lambda f ((\Psi(B)) f) K = 0 = 0 : K$.

Rules A_l

- $(M + N) V \rightarrow_\ell (M) V + (N) V$, with V being a value. Then $(M + N) V : K = (M) V + (N) V : K$.
- $(\alpha.M) V \rightarrow_\ell \alpha.(M) V$, with V being a value. Then $(\alpha.M) V : K = \alpha.(M) V : K$.
- $(0) V \rightarrow_\ell 0$, with V a value. Then $(0) V : K = 0 = 0 : K$.

Rules F and S

- $\alpha.(M + N) \rightarrow_\ell \alpha.M + \alpha.N$. Then $\alpha.(M + N) : K = \alpha.(M : K + N : K) \rightarrow_a \alpha.(M : K) + \alpha.(N : K) = \alpha.M + \alpha.N : K$.
- $\alpha.M + \beta.M \rightarrow_\ell (\alpha + \beta).M$. Then $\alpha.M + \beta.M : K = \alpha.(M : K) + \beta.(M : K) \rightarrow_a (\alpha + \beta).(M : K) = (\alpha + \beta).M : K$.

- $\alpha.M + M \rightarrow_\ell (\alpha + 1).M$. Then $\alpha.M + M : K = \alpha.M : K + M : K = \alpha.(M : K) + M : K \rightarrow_a (\alpha + 1).(M : K) = (\alpha + 1).M : K$.
- $M + M \rightarrow_\ell (1 + 1).M$. Then $M + M : K = M : K + M : K \rightarrow_a (1 + 1).(M : K) = (1 + 1).M : K$.
- $0 + M \rightarrow_\ell M$. Then $0 + M : K = (0 : K) + (M : K) = 0 + (M : K) \rightarrow_a M : K$.
- $\alpha.(\beta.M) \rightarrow_\ell (\alpha\beta).M$. Then $\alpha.(\beta.M) : K = \alpha.(\beta.M : K) = \alpha.(\beta.(M : K))$ which \rightarrow_a -reduces to $(\alpha\beta).(M : K) = (\alpha\beta).M : K$.
- $1.M \rightarrow_\ell M$. Then $1.M : K = 1.(M : K) \rightarrow_a M : K$.
- $0.M \rightarrow_\ell 0$. Then $0.M : K = 0.(M : K) \rightarrow_a 0 = 0 : K$.
- $\alpha.0 \rightarrow_\ell 0$. Then $\alpha.0 : K = \alpha.(0 : K) = \alpha.0 \rightarrow_a 0 = 0 : K$.

Rules *Asso* and *Com*

- $M + (N + L) \rightarrow_\ell (M + N) + L$. Then $M + (N + L) : K = M : K + (N + L : K) = M : K + (N : K + L : K) \rightarrow_a (M : K + N : K) + L : K = M + N : K + L : K = (M + N) + L : K$.
- $M + N \rightarrow_\ell N + M$. Then $M + N : K = M : K + N : K \rightarrow_a N : K + M : K = N + M : K$.

Rules ξ and $\xi_{\lambda_{in}}$ Assume $M \rightarrow_\ell M'$, and assume that for all K base term, $M : K \rightarrow_a^* M' : K$. We show that the result also holds for each contextual rule.

- $M + N \rightarrow_\ell M' + N$. Then $M + N : K = M : K + N : K \rightarrow_a^* M' : K + N : K = M' + N : K$.
- $N + M \rightarrow_\ell N + M'$, analogous to previous case.
- $\alpha.M \rightarrow_\ell \alpha.M'$. Then $\alpha.M : K = \alpha.(M : K) \rightarrow_a^* \alpha.(M' : K) = \alpha.M' : K$.
- $(V) M \rightarrow_\ell (V) M'$. Case by case:
 - $V = B$. Then $(B) M : K = M : \lambda f ((\Psi(B)) f) K$ which \rightarrow_a -reduces by the induction hypothesis to $M' : \lambda f ((\Psi(B)) f) K = (B) M' : K$.
 - $V = 0$. Then $(0) M : K = 0 = (0) M' : K$.
 - $V = \alpha.W$. Then $(\alpha.W) M : K = \alpha.(W) M : K = \alpha.((W) M : K)$ which \rightarrow_a -reduces by the induction hypothesis to $\alpha.((W) M' : K) = \alpha.(W) M' : K$.
 - $V = V_1 + V_2$. Then $(V_1 + V_2) M : K = (V_1) M + (V_2) M : K = (V_1) M : K + (V_2) M : K$ which \rightarrow_a -reduces by the induction hypothesis to $(V_1) M' : K + (V_2) M' : K = (V_1) M' + (V_2) M' : K = (V_1 + V_2) M' : K$.
- $(M) N \rightarrow_\ell (M') N$ Case by case:
 - $M = B$. Absurd since a base term cannot reduce.
 - $M = \alpha.M_1$. Case by case on the possible \rightarrow_ℓ -reductions of M :
 - * $M' = \alpha.M'_1$ with $M_1 \rightarrow_\ell M'_1$. Then $(\alpha.M_1) N : K = \alpha.(M_1) N : K = \alpha.((M_1) N : K)$ which by the induction hypothesis \rightarrow_a -reduces to $\alpha.((M'_1) N : K) = \alpha.(M'_1) N : K = (\alpha.M'_1) N : K$.
 - * $M = \alpha.(\beta.M_3)$ and $M' = (\alpha\beta).M_3$. Then $(\alpha.(\beta.M_3)) N : K = \alpha.(\beta.((M_3) N : K)) \rightarrow_a (\alpha\beta).((M_3) N : K) = ((\alpha\beta).M_3) N : K$.
 - * $M = \alpha.(L_1 + L_2)$ and $M' = \alpha.L_1 + \alpha.L_2$. Then $(\alpha.(L_1 + L_2)) N : K = \alpha.((L_1) N : K + (L_2) N : K) \rightarrow_a \alpha.((L_1) N : K) + \alpha.((L_2) N : K) = (\alpha.L_1 + \alpha.L_2) N : K$.

- * $\alpha = 1$ and $M' = M_1$. Then $(1.M_1) N : K = 1.((M_1) N : K) \rightarrow_a (M_1) N : K$.
- * $\alpha = 0$ and $M' = 0$. Then $(0.M_1) N : K = 0.((M_1) N : K) \rightarrow_a 0 = (0) N : K$.
- * $M_1 = 0$ and $M' = 0$. Then $(\alpha.0) N : K = \alpha.((0) N : K) = \alpha.0 \rightarrow_a 0 = (0) N : K$.
- $M = M_1 + M_2$. Case by case on the possible \rightarrow_ℓ -reductions of M :
 - * $M' = M'_1 + M_2$ with $M_1 \rightarrow_\ell M'_1$. Then $(M_1 + M_2) N : K = (M_1) N : K + (M_2) N : K$ which by the induction hypothesis \rightarrow_a -reduces to $(M'_1) N : K + (M_2) N : K = (M'_1 + M_2) N : K$.
 - * $M' = M_1 + M'_2$ with $M_2 \rightarrow_\ell M'_2$. Analogous to previous case.
 - * $M_2 = L_1 + L_2$ and $M' = (M_1 + L_1) + L_2$. Then $(M_1 + (L_1 + L_2)) N : K = (M_1) N : K + ((L_1) N : K + (L_2) N : K)$ and this \rightarrow_a -reduces to $((M_1) N : K + (L_1) N : K) + (L_2) N : K = ((M_1 + L_1) + L_2) N : K$.
 - * $M_1 = L_1 + L_2$ and $M' = L_1 + (L_2 + M_2)$. Analogous to previous case.
 - * $M' = M_2 + M_1$. Then $(M_1 + M_2) N : K = (M_1) N : K + (M_2) N : K \rightarrow_a (M_2) N : K + (M_1) N : K = (M_2 + M_1) N : K$.
 - * $M_1 = \alpha.M_3$, $M_2 = \beta.M_3$ and $M' = (\alpha + \beta).M_3$. Then $(\alpha.M_3 + \beta.M_3) N : K = \alpha.((M_3) N : K) + \beta.((M_3) N : K) \rightarrow_a (\alpha + \beta).((M_3) N : K) = ((\alpha + \beta).M_3) N : K$.
 - * $M_1 = \alpha.M_3$, $M_2 = M_3$ and $M' = (\alpha + 1).M_3$. Analogous to previous case.
 - * $M_1 = M_2$ and $M' = (1 + 1).M_1$. Analogous to previous case.
- $M = 0$. Absurd since 0 does not reduce.
- $M = (M_1) M_2$. Then the term $((M_1) M_2) N : K$ is equal to $(M_1) M_2 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$, which by the induction hypothesis \rightarrow_a -reduces to $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$. We do a second induction, over M' , to prove that $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K \rightarrow_a (M') N : K$.
 - * If $M' = (M'_1) M'_2$, then $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ is equal to $((M'_1) M'_2) N : K = (M') N : K$.
 - * M' cannot be a base term since from $(M_1) M_2$ it is not possible to arrive to a base term using only \rightarrow_ℓ .
 - * If $M' = \alpha.M'_1$, then $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K = \alpha.M'_1 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K = \alpha.(M'_1 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K)$ which \rightarrow_a -reduces by the induction hypothesis to $\alpha.((M'_1) N : K) = (\alpha.M'_1) N : K = (M') N : K$.
 - * If $M' = M'_1 + M'_2$, then the term $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ is equal to $M'_1 + M'_2 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ which is equal to $M'_1 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K + M'_2 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ which \rightarrow_a -reduces by the induction hypothesis to $(M'_1) N : K + (M'_2) N : K = (M'_1 + M'_2) N : K = (M') N : K$.
 - * If $M' = 0$ then $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ is equal to $0 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K = 0 = (0) N : K = (M') N : K$.

A.5 Proof of Lemma 6

Proof. Case by case on the rules of λ_{lin} .

Rule β_v

$$\begin{aligned}
 (\lambda x \ M) \ B : K &= B : \lambda f ((\Psi(\lambda x \ M)) \ f) \ K \\
 &= (\lambda f ((\Psi(\lambda x \ M)) \ f) \ K) \ \Psi(B) \\
 &\rightarrow_{\beta_n} ((\Psi(\lambda x \ M)) \ \Psi(B)) \ K \\
 &= ((\lambda x \ \llbracket M \rrbracket) \ \Psi(B)) \ K \\
 &\rightarrow_{\beta_n} \llbracket M \rrbracket[x := \Psi(B)] \ K \\
 (\text{Lemma 3}) &= \llbracket M[x := B] \rrbracket \ K \\
 (\text{Lemma 4}) &\rightarrow_{a\cup\beta}^* M[x := B] : K
 \end{aligned}$$

Algebraic rules If $M \rightarrow_\ell N$, then by Lemma 5 $M : K \rightarrow_a^* N : K$ which implies that $M : K \rightarrow_{a\cup\beta}^* N : K$.

Rules ξ and $\xi_{\lambda_{lin}}$ If $M \rightarrow_\ell M'$, then we use Lemma 5 to close the case. Assume $M \rightarrow_{\beta_v} M'$, and assume that for all K base term, $M : K \rightarrow_{a\cup\beta}^* M' : K$. We show that the result also holds for each contextual rule.

- $M + N \rightarrow_{\beta_v} M' + N$. Then $M + N : K = M : K + N : K \rightarrow_{a\cup\beta}^* M' : K + N : K = M' + N : K$.
- $N + M \rightarrow_{\beta_v} N + M'$, analogous to previous case.
- $\alpha.M \rightarrow_{\beta_v} \alpha.M'$. Then $\alpha.M : K = \alpha.(M : K) \rightarrow_{a\cup\beta}^* \alpha.(M' : K) = \alpha.M' : K$.
- $(V) \ M \rightarrow_{\beta_v} (V) \ M'$. Case by case:
 - $V = B$. Then $(B) \ M : K = M : \lambda f ((\Psi(B)) \ f) \ K$ which $\rightarrow_{a\cup\beta}$ -reduces by the induction hypothesis to $M' : \lambda f ((\Psi(B)) \ f) \ K = (B) \ M' : K$.
 - $V = 0$. Then $(0) \ M : K = 0 = (0) \ M' : K$.
 - $V = \alpha.W$. Then $(\alpha.W) \ M : K = \alpha.(W) \ M : K = \alpha.((W) \ M : K)$ which $\rightarrow_{a\cup\beta}$ -reduces by the induction hypothesis to $\alpha.((W) \ M' : K) = \alpha.(W) \ M' : K = (\alpha.W) \ M' : K$.
 - $V = V_1 + V_2$. Then $(V_1 + V_2) \ M : K = (V_1) \ M + (V_2) \ M : K = (V_1) \ M : K + (V_2) \ M : K$ which $\rightarrow_{a\cup\beta}$ -reduces by the induction hypothesis to $(V_1) \ M' : K + (V_2) \ M' : K = (V_1) \ M' + (V_2) \ M' : K = (V_1 + V_2) \ M' : K$.
- $(M) \ N \rightarrow_{\beta_v} (M') \ N$ Case by case:
 - $M = B$. Absurd since a base term cannot reduce.
 - $M = \alpha.M_1$. The only possible \rightarrow_{β_v} -reduction from M is $M' = \alpha.M'_1$ with $M_1 \rightarrow_{\beta_v} M'_1$. Then $(\alpha.M_1) \ N : K = \alpha.(M_1) \ N : K = \alpha.((M_1) \ N : K)$ which by the induction hypothesis $\rightarrow_{a\cup\beta}$ -reduces to $\alpha.((M'_1) \ N : K) = \alpha.(M'_1) \ N : K = (\alpha.M'_1) \ N : K$.
 - $M = M_1 + M_2$. Case by case on the possible \rightarrow_{β_v} -reductions of M :
 - * $M' = M'_1 + M_2$ with $M_1 \rightarrow_{\beta_v} M'_1$. Then $(M_1 + M_2) \ N : K = (M_1) \ N : K + (M_2) \ N : K$ which by the induction hypothesis $\rightarrow_{a\cup\beta}$ -reduces to $(M'_1) \ N : K + (M_2) \ N : K = (M'_1 + M_2) \ N : K$.

- * $M' = M_1 + M'_2$ with $M_2 \rightarrow_{\beta_v} M'_2$. Analogous to previous case.
- $M = 0$. Absurd since 0 does not reduce.
- $M = (M_1) M_2$. Then the term $((M_1) M_2) N : K$ is equal to $(M_1) M_2 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$, which $\rightarrow_{a \cup \beta}$ -reduces, by the induction hypothesis, to $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$. We do a second induction, over M' , to prove that $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K \rightarrow_{a \cup \beta}$ -reduces to $(M') N : K$.
 - * If $M' = (M'_1) M'_2$, then $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ is equal to $((M'_1) M'_2) N : K = (M') N : K$.
 - * If M' is a base term, then the term $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ is equal to $(\lambda g(\llbracket N \rrbracket) \lambda h((g) h) K) \Psi(M')$ which $\rightarrow_{a \cup \beta}$ -reduces to $(\llbracket N \rrbracket) \lambda h((\Psi(M')) h) K$ which, by Lemma 4, $\rightarrow_{a \cup \beta}$ -reduces to $N : \lambda h((\Psi(M')) h) K = (M') N : K$.
 - * If $M' = \alpha.M'_1$, then $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K = \alpha.M'_1 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K = \alpha.(M'_1 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K)$ which $\rightarrow_{a \cup \beta}$ -reduces by the induction hypothesis to $\alpha.((M'_1) N : K) = (\alpha.M'_1) N : K = (M') N : K$.
 - * If $M' = M'_1 + M'_2$, then the term $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ is equal to $M'_1 + M'_2 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ which is equal to $M'_1 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K + M'_2 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ which $\rightarrow_{a \cup \beta}$ -reduces by the induction hypothesis to $(M'_1) N : K + (M'_2) N : K = (M'_1 + M'_2) N : K = (M') N : K$.
 - * If $M' = 0$ then $M' : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K$ is equal to $0 : \lambda g(\llbracket N \rrbracket) \lambda h((g) h) K = 0 = (0) N : K = (M') N : K$.

A.6 Proof of Lemma 9

Proof. Structural induction on M .

- $M = x$. Then $\{x[x := N]\} = \{N\} = x[x := \{N\}] = \{x\}[x := \{N\}]$.
- $M = y$. Then $\{y[x := N]\} = y = \{y\}[x := \{N\}]$.
- $M = 0$. Analogous to previous case.
- $M = \lambda y M'$. Then

$$\begin{aligned}
 \{(\lambda y M')[x := N]\} &= \{\lambda y (M'[x := N])\} \\
 &= \lambda f (f) \lambda y \{M'[x := N]\} \\
 &\quad \text{by the induction hypothesis} \\
 &= \lambda f (f) \lambda y \{M'\}[x := \{N\}] \\
 &= (\lambda f (f) \lambda y \{M'\})[x := \{N\}] \\
 &= \{M\}[x := \{N\}]
 \end{aligned}$$

- $M = (N_1) N_2$. Then

$$\begin{aligned}
 \{M[x := N]\} &= \{((N_1) N_2)[x := N]\} \\
 &= \{(N_1[x := N]) N_2[x := N]\}
 \end{aligned}$$

$$\begin{aligned}
&= \lambda f (\llbracket N_1[x := N] \rrbracket) \lambda g ((g) \llbracket N_2[x := N] \rrbracket) f \\
&\quad \text{by the induction hypothesis} \\
&= \lambda f (\llbracket N_1 \rrbracket[x := \llbracket N \rrbracket]) \lambda g ((g) \llbracket N_2 \rrbracket[x := \llbracket N \rrbracket]) f \\
&= (\lambda f (\llbracket N_1 \rrbracket) \lambda g ((g) \llbracket N_2 \rrbracket) f)[x := \llbracket N \rrbracket] \\
&= \llbracket (N_1) N_2 \rrbracket[x := \llbracket N \rrbracket] \\
&= \llbracket M \rrbracket[x := \llbracket N \rrbracket]
\end{aligned}$$

– $M = \alpha.M'$. Then

$$\begin{aligned}
\llbracket M[x := N] \rrbracket &= \llbracket (\alpha.M')[x := N] \rrbracket \\
&= \llbracket \alpha.(M'[x := N]) \rrbracket \\
&= \lambda f (\alpha.\llbracket M'[x := N] \rrbracket) f
\end{aligned}$$

by the induction hypothesis

$$\begin{aligned}
&= \lambda f (\alpha.\llbracket M' \rrbracket[x := \llbracket N \rrbracket]) f \\
&= \lambda f (\alpha.\llbracket M' \rrbracket) f[x := \llbracket N \rrbracket] \\
&= \llbracket \alpha.M' \rrbracket[x := \llbracket N \rrbracket] \\
&= \llbracket M \rrbracket[x := \llbracket N \rrbracket]
\end{aligned}$$

– $M = N_1 + N_2$. Then

$$\begin{aligned}
\llbracket M[x := N] \rrbracket &= \llbracket (N_1 + N_2)[x := N] \rrbracket \\
&= \llbracket N_1[x := N] + N_2[x := N] \rrbracket \\
&= \lambda f (\llbracket N_1[x := N] \rrbracket + \llbracket N_2[x := N] \rrbracket) f \\
&\quad \text{by the induction hypothesis} \\
&= \lambda f (\llbracket N_1 \rrbracket[x := \llbracket N \rrbracket] + \llbracket N_2 \rrbracket[x := \llbracket N \rrbracket]) f \\
&= \lambda f ((\llbracket N_1 \rrbracket + \llbracket N_2 \rrbracket) f[x := \llbracket N \rrbracket]) \\
&= \llbracket N_1 + N_2 \rrbracket[x := \llbracket N \rrbracket] \\
&= \llbracket M \rrbracket[x := \llbracket N \rrbracket]
\end{aligned}$$

A.7 Proof of Lemma 10

Proof. Structural induction on M .

- $M = \lambda x N$. Then $(\llbracket \lambda x N \rrbracket) K = (\lambda f (f) \lambda x \llbracket N \rrbracket) K$ and by definition of Φ this is equal to $(\lambda f (f) \Phi(\lambda x N)) K \rightarrow_{\ell \cup \beta} (K) \Phi(\lambda x N) = \lambda x N : K$.
- $M = 0$. Then $(\llbracket 0 \rrbracket) K = (\lambda f (0) f) K \rightarrow_{\ell \cup \beta} (0) K \rightarrow_{\ell \cup \beta} 0 = 0 : K$.
- $M = M' + N$. Then $(\llbracket M' + N \rrbracket) K = (\lambda f (\llbracket M' \rrbracket + \llbracket N \rrbracket) f) K \rightarrow_{\ell \cup \beta} (\llbracket M' \rrbracket + \llbracket N \rrbracket) K$ which $\rightarrow_{\ell \cup \beta}$ -reduces by the induction hypothesis to $M' : K + N : K = M' + N : K$.
- $M = \alpha.N$. Then $(\llbracket \alpha.N \rrbracket) K = (\lambda f (\alpha.\llbracket N \rrbracket) f) K \rightarrow_{\ell \cup \beta} (\alpha.\llbracket N \rrbracket) K$ which $\rightarrow_{\ell \cup \beta}$ -reduces to $\alpha.(\llbracket N \rrbracket) K$ and this, by the induction hypothesis, $\rightarrow_{\ell \cup \beta}$ -reduces to $\alpha.(N : K) = \alpha.N : K$.

- $M = (M') N$. Then $(\llbracket (M') N \rrbracket) K = (\lambda f (\llbracket M' \rrbracket) \lambda g ((g) \llbracket N \rrbracket) f) K$ which $\rightarrow_{\ell \cup \beta}$ -reduces to $(\llbracket M' \rrbracket) \lambda g ((g) \llbracket N \rrbracket) K$. Note that $\lambda g ((g) \llbracket N \rrbracket) K$ is a value, so by the induction hypothesis the above term reduces to $M' : \lambda g ((g) \llbracket N \rrbracket) K$. We do a second induction, over M' , to prove that $M' : \lambda g ((g) \llbracket N \rrbracket) K \rightarrow_{\ell \cup \beta}^* (M') N : K$.
 - If $M' = (M_1) M_2$, then $M' : \lambda g ((g) \llbracket N \rrbracket) K = ((M_1) M_2) N : K = (M') N : K$.
 - If M' is a base term, $M' : \lambda g ((g) \llbracket N \rrbracket) K = (\lambda g ((g) \llbracket N \rrbracket) K) \Phi(M')$ which $\rightarrow_{\ell \cup \beta}$ -reduces to $((\Phi(M')) \llbracket N \rrbracket) K = (M') N : K$.
 - If $M' = \alpha.M_1$, then $\alpha.M_1 : \lambda g ((g) \llbracket N \rrbracket) K = \alpha.(M_1 : \lambda g ((g) \llbracket N \rrbracket) K)$ which $\rightarrow_{\ell \cup \beta}^*$ -reduces by the induction hypothesis to $\alpha.((M_1) N : K) = (\alpha.M_1) N : K = (M') N : K$.
 - If $M' = M_1 + M_2$, then $M' : \lambda g ((g) \llbracket N \rrbracket) K = M_1 + M_2 : \lambda g ((g) \llbracket N \rrbracket) K$ which is equal to $M_1 : \lambda g ((g) \llbracket N \rrbracket) K + M_2 : \lambda g ((g) \llbracket N \rrbracket) K$ which $\rightarrow_{\ell \cup \beta}^*$ -reduces by the induction hypothesis to $(M_1) N : K + (M_2) N : K = (M_1 + M_2) N : K = (M') N : K$.
 - If $M' = 0$ then $M' : \lambda g ((g) \llbracket N \rrbracket) K = 0 : \lambda g ((g) \llbracket N \rrbracket) K = 0 = (0) N : K = (M') N : K$.

A.8 Proof of Lemma 11

Proof. Case by case on the rules of λ_{alg} .

Rule β_v

$$\begin{aligned}
 (\lambda x M) N : K &= ((\Phi(\lambda x M)) \llbracket N \rrbracket) K \\
 &= ((\lambda x (\llbracket M \rrbracket)) \llbracket N \rrbracket) K \\
 \text{(since } \llbracket N \rrbracket \text{ is a base term)} &\rightarrow_{\ell \cup \beta} \llbracket M \rrbracket[x := (\llbracket N \rrbracket)] K \\
 \text{(Lemma 9)} &= \llbracket M[x := N] \rrbracket K \\
 \text{(Lemma 10)} &\rightarrow_{\ell \cup \beta}^* M[x := N] : K
 \end{aligned}$$

Rules A – Let $(M + N) L \rightarrow_{a \cup \beta} (M) L + (N) L$. $(M + N) L : K = ((M) L + (N) L) : K$.

- Let $(\alpha.M) N \rightarrow_{a \cup \beta} \alpha.(M) N$. $(\alpha.M) N : K = \alpha.(M) N : K$
- Let $(0) N \rightarrow_{a \cup \beta} 0$. $(0) N : K = 0 = 0 : K$

Rules F and S

- $\alpha.(M + N) \rightarrow_{a \cup \beta} \alpha.M + \alpha.N$. Then $\alpha.(M + N) : K = \alpha.(M : K + N : K) \rightarrow_{\ell \cup \beta} \alpha.(M : K) + \alpha.(N : K) = \alpha.M + \alpha.N : K$.
- $\alpha.M + \beta.M \rightarrow_{a \cup \beta} (\alpha + \beta).M$. Then $\alpha.M + \beta.M : K = \alpha.(M : K) + \beta.(M : K) \rightarrow_{\ell \cup \beta} (\alpha + \beta).(M : K) = (\alpha + \beta).M : K$.
- $\alpha.M + M \rightarrow_{a \cup \beta} (\alpha + 1).M$. Then $\alpha.M + M : K = \alpha.M : K + M : K = \alpha.(M : K) + M : K \rightarrow_{\ell \cup \beta} (\alpha + 1).(M : K) = (\alpha + 1).M : K$.
- $M + M \rightarrow_{a \cup \beta} (1 + 1).M$. Then $M + M : K = M : K + M : K \rightarrow_{\ell \cup \beta} (1 + 1).(M : K) = (1 + 1).M : K$.
- $0 + M \rightarrow_{a \cup \beta} M$. Then $0 + M : K = (0 : K) + (M : K) = 0 + (M : K) \rightarrow_{\ell \cup \beta} M : K$.

- $\alpha.(\beta.M) \rightarrow_{a\cup\beta} (\alpha\beta).M$. Then $\alpha.(\beta.M) : K = \alpha.(\beta.M : K) = \alpha.(\beta.(M : K))$ which $\rightarrow_{\ell\cup\beta}$ -reduces to $(\alpha\beta).(M : K) = (\alpha\beta).M : K$.
- $1.M \rightarrow_{a\cup\beta} M$. Then $1.M : K = 1.(M : K) \rightarrow_{\ell\cup\beta} M : K$.
- $0.M \rightarrow_{a\cup\beta} 0$. Then $0.M : K = 0.(M : K) \rightarrow_{\ell\cup\beta} 0 = 0 : K$.
- $\alpha.0 \rightarrow_{a\cup\beta} 0$. Then $\alpha.0 : K = \alpha.(0 : K) = \alpha.0 \rightarrow_{\ell\cup\beta} 0 = 0 : K$.

Rules Asso and Com

- $M + (N + L) \rightarrow_{a\cup\beta} (M + N) + L$. Then $M + (N + L) : K = M : K + (N + L : K) = M : K + (N : K + L : K) \rightarrow_{\ell\cup\beta} (M : K + N : K) + L : K = M + N : K + L : K = (M + N) + L : K$.
- $M + N \rightarrow_{a\cup\beta} N + M$. Then $M + N : K = M : K + N : K \rightarrow_{\ell\cup\beta} N : K + M : K = N + M : K$.

Rules ξ Assume $M \rightarrow_{a\cup\beta} M'$, and that for all K base term, $M : K \rightarrow_{\ell\cup\beta}^* M' : K$. We show that the result also holds for each contextual rule.

- $M + N \rightarrow_{a\cup\beta} M' + N$. Then $M + N : K = M : K + N : K \rightarrow_{\ell\cup\beta}^* M' : K + N : K = M' + N : K$.
- $N + M \rightarrow_{a\cup\beta} N + M'$, analogous to previous case.
- $\alpha.M \rightarrow_{a\cup\beta} \alpha.M'$. Then $\alpha.M : K = \alpha.(M : K) \rightarrow_{\ell\cup\beta}^* \alpha.(M' : K) = \alpha.M' : K$.
- $(M) N \rightarrow_{a\cup\beta} (M') N$ Case by case:
 - $M = B$. Absurd since a base term cannot reduce.
 - $M = \alpha.M_1$. Case by case on the possible $\rightarrow_{a\cup\beta}$ -reductions of M :
 - * $M' = \alpha.M'_1$ with $M_1 \rightarrow_{a\cup\beta} M'_1$. Then $(\alpha.M_1) N : K = \alpha.(M_1) N : K = \alpha.((M_1) N : K)$ which by the induction hypothesis $\rightarrow_{\ell\cup\beta}$ -reduces to $\alpha.((M'_1) N : K) = \alpha.(M'_1) N : K = (\alpha.M'_1) N : K$.
 - * $M = \alpha.(\beta.M_3)$ and $M' = (\alpha\beta).M_3$. Then $(\alpha.(\beta.M_3)) N : K = \alpha.(\beta.((M_3) N : K)) \rightarrow_{\ell\cup\beta} (\alpha\beta).((M_3) N : K) = ((\alpha\beta).M_3) N : K$.
 - * $M = \alpha.(L_1 + L_2)$ and $M' = \alpha.L_1 + \alpha.L_2$. Then $(\alpha.(L_1 + L_2)) N : K = \alpha.((L_1) N : K + (L_2) N : K) \rightarrow_{\ell\cup\beta} \alpha.((L_1) N : K) + \alpha.((L_2) N : K) = (\alpha.L_1 + \alpha.L_2) N : K$.
 - * $\alpha = 1$ and $M' = M_1$. Then $(1.M_1) N : K = 1.((M_1) N : K) \rightarrow_{\ell\cup\beta} (M_1) N : K$.
 - * $\alpha = 0$ and $M' = 0$. Then $(0.M_1) N : K = 0.((M_1) N : K) \rightarrow_{\ell\cup\beta} 0 = (0) N : K$.
 - * $M_1 = 0$ and $M' = 0$. Then $(\alpha.0) N : K = \alpha.((0) N : K) = \alpha.0 \rightarrow_{\ell\cup\beta} 0 = (0) N : K$.
 - $M = M_1 + M_2$. Case by case on the possible $\rightarrow_{a\cup\beta}$ -reductions of M :
 - * $M' = M'_1 + M_2$ with $M_1 \rightarrow_{a\cup\beta} M'_1$. Then $(M_1 + M_2) N : K = (M_1) N : K + (M_2) N : K$ which by the induction hypothesis $\rightarrow_{\ell\cup\beta}$ -reduces to $(M'_1) N : K + (M_2) N : K = (M'_1 + M_2) N : K$.
 - * $M' = M_1 + M'_2$ with $M_2 \rightarrow_{a\cup\beta} M'_2$. Analogous to previous case.
 - * $M_2 = L_1 + L_2$ and $M' = (M_1 + L_1) + L_2$. Then $(M_1 + (L_1 + L_2)) N : K = (M_1) N : K + ((L_1) N : K + (L_2) N : K)$ which $\rightarrow_{\ell\cup\beta}$ -reduces to $((M_1) N : K + (L_1) N : K) + (L_2) N : K = ((M_1 + L_1) + L_2) N : K$.

- * $M_1 = L_1 + L_2$ and $M' = L_1 + (L_2 + M_2)$. Analogous to previous case.
- * $M' = M_2 + M_1$. Then $(M_1 + M_2) N : K = (M_1) N : K + (M_2) N : K \rightarrow_{\ell \cup \beta} (M_2) N : K + (M_1) N : K = (M_2 + M_1) N : K$.
- * $M_1 = \alpha.M_3$, $M_2 = \beta.M_3$ and $M' = (\alpha + \beta).M_3$. Then $(\alpha.M_3 + \beta.M_3) N : K = \alpha.((M_3) N : K) + \beta.((M_3) N : K) \rightarrow_{\ell \cup \beta} (\alpha + \beta).((M_3) N : K) = ((\alpha + \beta).M_3) N : K$.
- * $M_1 = \alpha.M_3$, $M_2 = M_3$ and $M' = (\alpha + 1).M_3$. Analogous to previous case.
- * $M_1 = M_2$ and $M' = (1 + 1).M_1$. Analogous to previous case.
- $M = 0$. Absurd since 0 does not reduce.
- $M = (M_1) M_2$. Then $((M_1) M_2) N : K$ is equal to $(M_1) M_2 : \lambda g((g) \{N\}) K$, which by the induction hypothesis $\rightarrow_{\ell \cup \beta}$ -reduces to $M' : \lambda g((g) \{N\}) K$. We do a second induction, over M' , to prove that $M' : \lambda g((g) \{N\}) K \rightarrow_{\ell \cup \beta}^* (M') N : K$.
 - * If $M' = (M'_1) M'_2$, then $M' : \lambda g((g) \{N\}) K = ((M'_1) M'_2) N : K = (M') N : K$.
 - * If M' is a base term, then $M' : \lambda g((g) \{N\}) K$ is equal to $(\lambda g((g) \{N\}) K) \Phi(M') \rightarrow_{\ell \cup \beta} ((\Phi(M')) \{N\}) K = (M') N : K$.
 - * If $M' = \alpha.M'_1$, then $\alpha.M'_1 : \lambda g((g) \{N\}) K$ is equal to $\alpha.(M'_1 : \lambda g((g) \{N\}) K)$ which $\rightarrow_{\ell \cup \beta}^*$ -reduces by the induction hypothesis to $\alpha.((M'_1) N : K) = (\alpha.M'_1) N : K = (M') N : K$.
 - * If $M' = M'_1 + M'_2$, then $M' : \lambda g((g) \{N\}) K = M'_1 + M'_2 : \lambda g((g) \{N\}) K$ which is equal to $M'_1 : \lambda g((g) \{N\}) K + M'_2 : \lambda g((g) \{N\}) K$ which $\rightarrow_{\ell \cup \beta}^*$ -reduces by the induction hypothesis to $(M'_1) N : K + (M'_2) N : K = (M'_1 + M'_2) N : K = (M') N : K$.
 - * If $M' = 0$ then $M' : \lambda g((g) \{N\}) K = 0 : \lambda g((g) \{N\}) K = 0 = (0) N : K = (M') N : K$

B COQ proof of 1

The proof of the local confluence of the algebraic fragments of $\lambda_{lin}^{\rightarrow}$ and $\lambda_{alg}^{\rightarrow}$ are sufficiently monotonous so that one can ask a proof assistant to do them. For this purpose we use the library `LocConf` setting up some convenient tactics. The interested reader can find the whole set of files in [23]:

- `RW.v`, `ListTac.v` and `LocConfTac.v` are the files containing the library;
- `Llin.v` and `Lalg.v` respectively contain the proofs for $\lambda_{lin}^{\rightarrow}$ and $\lambda_{alg}^{\rightarrow}$.

To compile the files, you will need COQ v.8.2pl1. and the `Ssreflect` extension v.1.2. Proceed with a flavour of:

```
$ coqc RW.v ListTac.v LocConfTac.v
$ coqc Llin.v
$ coqc Lalg.v
```

To check that no particular assumption were made, you can use

```
$ coqchk -o Llin
$ coqchk -o Lalg
```

B.1 Summary of the proof.

We summarise the content of `Llin.v` and `Lalg.v`.

Let us first define the set of scalars.

```
Variable scalar : Set.

Variable Sadd : scalar -> scalar -> scalar.
Variable Smul : scalar -> scalar -> scalar.
Variable S0 : scalar.
Variable S1 : scalar.

Notation "A + B" := (Sadd A B) : scalar_scope.
Notation "A * B" := (Smul A B) : scalar_scope.

Open Scope scalar_scope.

Hypothesis S_0_1_dec : ~ S1 = S0.
Hypothesis S_0_lunit : forall a, S0 + a = a.
Hypothesis S_0_lelim : forall a, S0 * a = S0.
Hypothesis S_1_lunit : forall a, S1 * a = a.
Hypothesis S_rdistrib : forall a b c, a*(b+c) = (a*b)+(a*c).
Hypothesis S_ldistrib : forall a b c, (a+b)*c = (a*c)+(b*c).
Hypothesis S_add_assoc : forall a b c, (a+b)+c = a+(b+c).
Hypothesis S_mul_assoc : forall a b c, (a*b)*c = a*(b*c).
Hypothesis S_add_commut : forall a b, a+b = b+a.
Hypothesis S_mul_commut : forall a b, a*b = b*a.

Close Scope scalar_scope.
```

We then define the set of terms. Values and bases are properties on terms defined by induction.

```
Inductive term : Set :=
| T0 : term
| Tadd : term -> term -> term
| Tmul : scalar -> term -> term
| Tvar : nat -> term
| Tlambda : term -> term
| Tapply : term -> term -> term.

Notation "A +s B" := (Sadd A B) (at level 50) : term_scope.
Notation "A *s B" := (Smul A B) (at level 40) : term_scope.
```

```

Notation "A + B" := (Tadd A B) : term_scope.
Notation "A '**' B" := (Tmul A B) (at level 35) : term_scope.
Notation "@ A" := (Tvar A) (at level 10) : term_scope.
Notation "A ; B" := (Tapply A B) (at level 30) : term_scope.
Notation "\ A" := (Tlambda A) (at level 40) : term_scope.

```

```

Open Scope term_scope.

```

```

Inductive is_value : term -> Prop :=
| valT0 : is_value T0
| valTlambda : forall s, is_value (Tlambda s)
| valTvar : forall n, is_value (Tvar n)
| valTmulbase : forall a s, is_value s -> is_value (a ** s)
| valTadd : forall s t,
  is_value s -> is_value t -> is_value (s + t).

```

```

Inductive is_base : term -> Prop :=
| baseTlambda : forall s, is_base (Tlambda s)
| baseTvar : forall n, is_base (Tvar n).

```

The definition of local confluence is given in the file `RW.v` of the library:

```

Section RW.

```

```

(** The relation is on some terms *)

```

```

Variable term : Set.

```

```

(** It is a binary proposition *)

```

```

Variable R : term -> term -> Prop.

```

```

(** Transitivity closure of the relation *)

```

```

Inductive Rstar : term -> term -> Prop :=
| Rzero : forall r, Rstar r r
| Rcons : forall r t s, (R r s) -> (Rstar s t) -> (Rstar r t).

```

```

Definition local_confluent :=
forall r s t,
R r s -> R r t ->
exists u, Rstar s u /\ Rstar t u.

```

```

End RW.

```

B.2 $\lambda_{lin}^{\rightarrow}$

We can now set up the rewrite system of $\lambda_{lin}^{\rightarrow}$. We use the notion of base for the right-linearity of the application.

Section Term.

Hypothesis R : term -> term -> Prop.

```
(** Elementary rules *)
Definition R_T0_runit := forall t, R (t + T0) t.
Definition R_S0_anni := forall t, R (S0 ** t) T0.
Definition R_S1_unit := forall t, R (S1 ** t) t.
Definition R_T0_anni := forall a, R (a ** T0) T0.
Definition R_mul_abs :=
  forall a b t, R (a ** (b ** t)) ((a *s b) ** t).
Definition R_ma_dist :=
  forall a s t, R (a ** (s + t)) (a ** s + a ** t).
(** Factorization *)
Definition R_add_fact :=
  forall a b t, R (a ** t + b ** t) ((a +s b) ** t).
Definition R_add_fact1 :=
  forall a t, R (a ** t + t) ((a +s S1) ** t).
Definition R_add_fact11 := forall t, R (t + t) ((S1 +s S1) ** t).
(** Assoc. and commut. of addition *)
Definition R_add_com := forall s t, R (s + t) (t + s).
Definition R_add_rassoc :=
  forall r s t, R ((r + s) + t) (r + (s + t)).
Definition R_add_lassoc :=
  forall r s t, R (r + (s + t)) ((r + s) + t).
(** Congruence *)
Definition R_cong_mul := forall a s t, R s t -> R (a**s) (a**t).
Definition R_cong_ladd := forall u s t, R s t -> R (s+u) (t+u).
Definition R_cong_radd := forall u s t, R s t -> R (u+ s) (u+t).
Definition R_cong_lapp := forall u s t, R s t -> R (s;u) (t;u).
Definition R_cong_rapp :=
  forall u s t, is_value u -> R s t -> R (u;s) (u;t).
(** Linearity of application *)
Definition R_add_app_ldist :=
  forall r s t, is_value t -> R ((r + s);t) (r;t + s;t).
Definition R_mul_app_ldist :=
  forall a r s, is_value s -> R ((a**r);s) (a**(r;s)).
Definition R_T0_app_ldist :=
  forall s, is_value s -> R (T0;s) T0.
Definition R_add_app_rdist :=
  forall r s t, is_base t -> R (t;(r + s)) (t;r + t;s).
```

```

Definition R_mul_app_rdist :=
  forall a r s, is_base s -> R (s;(a**r)) (a**(s;r)).
Definition R_T0_app_rdist :=
  forall s, is_base s -> R (s;T0) T0.

End Term.

```

```

Inductive R : term -> term -> Prop :=
| ax1 :R_T0_runit R
| ax2 :R_S0_anni R
| ax3 :R_S1_unit R
| ax4 :R_T0_anni R
| ax5 :R_mul_abs R
| ax6 :R_ma_dist R

| ax7 :R_add_fact R
| ax8 :R_add_fact1 R
| ax9 :R_add_fact11 R

| ax10 :R_add_com R
| ax11 :R_add_rassoc R
| ax12 :R_add_lassoc R

| ax13 :R_cong_mul R
| ax14 :R_cong_ladd R
| ax15 :R_cong_radd R
| ax16 :R_cong_lapp R
| ax17 :R_cong_rapp R

| ax18 :R_add_app_ldist R
| ax19 :R_mul_app_ldist R
| ax20 :R_T0_app_ldist R
| ax21 :R_add_app_rdist R
| ax22 :R_mul_app_rdist R
| ax23 :R_T0_app_rdist R.

```

The theorem stating the local confluence of R reads as follows:

```

Theorem R_local_confluence : forall r s t:term,
(R r s) -> (R r t) ->
  exists u:term, (Rstar R s u) /\ (Rstar R t u).

```

B.3 $\lambda_{alg}^{\rightarrow}$

The rewrite system of $\lambda_{alg}^{\rightarrow}$ is simpler, since it does not consider values.

Section Term.

Hypothesis R : term -> term -> Prop.

```
(** Elementary rules *)
Definition R_T0_runit := forall t, R (t + T0) t.
Definition R_S0_anni := forall t, R (S0 ** t) T0.
Definition R_S1_unit := forall t, R (S1 ** t) t.
Definition R_T0_anni := forall a, R (a ** T0) T0.
Definition R_mul_abs :=
  forall a b t, R (a ** (b ** t)) ((a *s b) ** t).
Definition R_ma_dist :=
  forall a s t, R (a ** (s + t)) (a ** s + a ** t).
(** Factorization *)
Definition R_add_fact :=
  forall a b t, R (a ** t + b ** t) ((a +s b) ** t).
Definition R_add_fact1 :=
  forall a t, R (a ** t + t) ((a +s S1) ** t).
Definition R_add_fact11 :=
  forall t, R (t + t) ((S1 +s S1) ** t).
(** Assoc. and commut. of addition *)
Definition R_add_com := forall s t, R (s + t) (t + s).
Definition R_add_rassoc :=
  forall r s t, R ((r + s) + t) (r + (s + t)).
Definition R_add_lassoc :=
  forall r s t, R (r + (s + t)) ((r + s) + t).
(** Congruence *)
Definition R_cong_mul := forall a s t, R s t -> R (a**s) (a**t).
Definition R_cong_ladd := forall u s t, R s t -> R (s+u) (t+u).
Definition R_cong_radd := forall u s t, R s t -> R (u+ s) (u+t).
Definition R_cong_lapp := forall u s t, R s t -> R (s;u) (t;u).
(** Linearity of application *)
Definition R_add_app_ldist :=
  forall r s t, R ((r + s);t) (r;t + s;t).
Definition R_mul_app_ldist :=
  forall a r s, R ((a**r);s) (a**(r;s)).
Definition R_T0_app_ldist := forall s, R (T0;s) T0.
```

End Term.

```
Inductive R : term -> term -> Prop :=
| ax1 :R_T0_runit R
| ax2 :R_S0_anni R
| ax3 :R_S1_unit R
| ax4 :R_T0_anni R
```



```

| ax5 :R_mul_abs R
| ax6 :R_ma_dist R

| ax7 :R_add_fact R
| ax8 :R_add_fact1 R
| ax9 :R_add_fact11 R

| ax10 :R_add_com R
| ax11 :R_add_rassoc R
| ax12 :R_add_lassoc R

| ax13 :R_cong_mul R
| ax14 :R_cong_ladd R
| ax15 :R_cong_radd R
| ax16 :R_cong_lapp R

| ax18 :R_add_app_ldist R
| ax19 :R_mul_app_ldist R
| ax20 :R_T0_app_ldist R.

```

The statement of local confluence for R is the same as in the previous section:

```

Theorem R_local_confluence : forall r s t:term,
(R r s) -> (R r t) ->
  exists u:term, (Rstar R s u) /\ (Rstar R t u).

```